June, 1979

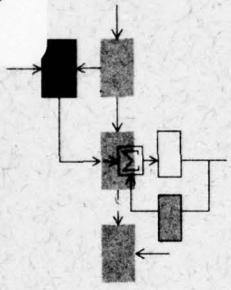
LIDS-TH-918

Research Supported By:

ARPA Contract ONR-N00014-75-C-1183

12

LEVEL



AN EFFICIENT CONTENTION RESOLUTION ALGORITHM FOR MULTIPLE ACCESS CHANNELS

Jeannine Mosely

Laboratory for Information and Decision Systems
Formerly
Electronic Systems Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY, CAMBRIDGE, MASSACHUSETTS 02139

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

UNCLASSIFIED

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. (9) 3. RECIPIENT'S CATALOG NUMBER: Master's thesis
4. TITLE (and Subtitle): (6) AN EFFICIENT CONTENTION RESOLUTION ALGORITHM FOR MULTIPLE ACCESS CHANNELS	5. TYPE OF REPORT & PERIOD COVERED: Technical
	6. PERFORMING ORG. REPORT NUMBER (14): LIDS-TH-918
7. AUTHOR(s): (10) Jeannine Mosely	8. CONTRACT OR GRANT NUMBER(s) (15): ARPA Order No. 3045/5-7-75; ONR N00014-75-C-1183; ARPA Order-3045
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Massachusetts Institute of Technology, Laboratory for Information and Decision Systems, Cambridge, Massachusetts 02139	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: Program Code No. 5T10, ONR Identifying No. 049-383
11. CONTROLLING OFFICE NAME AND ADDRESS: Defense Advanced Research Projects Agency, 1400 Wilson Boulevard, Arlington, Virginia 22209	12. REPORT DATE: May 1979 (11) Jun 79; 13. NUMBER OF PAGES: 81
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Office of Naval Research, Information Systems Program, Code 437, Arlington, Virginia 22217 (12) 84p	15. SECURITY CLASS. (of this report): UNCLASSIFIED; 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Multiaccess Channel ALOHA Random-Access Channel

410 950

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

When independent sources use a common broadcast channel, interference between the sources can cause messages to be lost. A common recovery procedure is to wait a random length of time before attempting retransmission. Capetanakis has devised a tree algorithm for determining when users should retransmit. This algorithm has better throughput, delay and stability characteristics than the random retransmission procedure. Gallager and Humblet have modified it to have higher throughput and the property that messages are transmitted in the order of generation.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

79 08 03 050

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. In this thesis, the general form of a first-come first-served contention resolution algorithm is given. A sub-class of FCFS algorithms, called single interval algorithms, are analyzed using Markovian decision theory. It is seen that, if the horizon is infinite, Humblet's algorithm has the maximum throughput for this sub-class. However, if the system is shut down after a finite number of transmissions, then an extended version of his algorithm is optimal.

May 1979 LIDS-TH- 918

AN EFFICIENT CONTENTION RESOLUTION ALGORITHM FOR MULTIPLE ACCESS CHANNELS

by

Jeannine Mosely

This report is based on the unaltered thesis of Jeannine Mosely, submitted in partial fulfillment of the requirements for the degree of Master of Science at the Massachusetts Institute of Technology, May 1979. The research was conducted at the M.I.T. Laboratory for Information and Decision Systems, with support provided in part by the Office of Naval Research under Contract ONR-N00014-75-C-1183.

Laboratory for Information and Decision Systems
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

AN EFFICIENT CONTENTION RESOLUTION ALGORITHM FOR MULTIPLE ACCESS CHANNELS

by

JEANNINE MOSELY

B.A., University of Illinois
(1974)

B.S., University of Illinois
(1977)

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE
DEGREE OF

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May, 1979

Signature of Author.....
Department of Electrical Engineering and Computer Science
May 25,1979

Certified by.....
Thesis Supervisor

Accepted by.....
Chairman, Departmental Committee on Graduate Students

AN EFFICIENT CONTENTION RESOLUTION ALGORITHM

FOR MULTIPLE ACCESS CHANNELS

by

Jeannine Mosely

Submitted to the Department of Electrical Engineering and Computer Science on May 25, 1979 in partial fulfillment of the requirements for the degree of Master of Science.

ABSTRACT

When independent sources use a common broadcast channel, interference between the sources can cause messages to be lost. A common recovery procedure is to wait a random length of time before attempting retransmission. Capetanakis has devised a tree algorithm for determining when users should retransmit. This algorithm has better throughput, delay and stability characteristics than the random retransmission procedure. Gallager and Humblet have modified it to have higher throughput and the property that messages are transmitted in the order of generation.

In this thesis, the general form of a first-come first-served contention resolution algorithm is given. A sub-class of FCFS algorithms, called single interval algorithms, are analyzed using Markovian decision theory. It is seen that, if the horizon is infinite, Humblet's algorithm has the maximum throughput for this sub-class. However, if the

system is shut down after a finite number of transmissions, then an extended version of his algorithm is optimal.

THESIS SUPERVISOR: Pierre A. Humblet

TITLE: Assistant Professor of Electrical Engineering

ACKNOWLEDGEMENTS

I would like to extend my sincere gratitude to Prof. Pierre A. Humblet for his help and encouragement throughout my research.

I would also like to thank Prof. Gallager, who first brought the problem to my attention.

I also thank Ms. Holly Glaser for her preparation of some of the diagrams in this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

In this thesis we present the most general form for a contention resolution algorithm that is first-come first-served. We model the algorithm as a Markov process with state transition rewards and use Markovian decision theory to analyze the maximum throughput, or capacity, for several restricted forms of the general algorithm.

In this chapter, we present the system model used. Existing algorithms are described and evaluated qualitatively. The general first-come first-served algorithm is given, and the sub-class of FCFS algorithms which we will analyze is described. We complete the chapter with an outline of the thesis.

1.1 Description of the Problem

We are concerned with a large number of bursty users trying to communicate with a central facility on a common channel. The traditional methods of multi-accessing, time division multi-accessing (TDMA) and frequency division multi-accessing (FDMA) are not suitable. Since the number of users is assumed to be very large, FDMA is prohibitive in hardware costs, and since the users are bursty as well, both FDMA and TDMA require trade-offs between delay and bandwidth that result in either unacceptable delays or inefficient use of channel capacity. In this situation schemes have been proposed in which the users access the

channel randomly and, when interference causes messages to be lost, a contention resolution algorithm is applied to determine when the users should retransmit.

There are several criteria for judging such a scheme. It should be stable; that is, the expected number of users waiting to transmit should remain small. It should allow messages to be transmitted with low average delay. And it should have a high throughput (percentage of capacity used to transmit successfully). Algorithms which meet these three requirements already exist and we will review some of them below.

1.2 Source and Channel Models

Before discussing the algorithms, we will describe the source and channel models used. We consider a central facility with an infinite number of users, also called sources. The sources generate messages of fixed length, referred to as packets. Because the number of users is infinite, each source is assumed to generate only one message. A slot is the length of time required to transmit a packet. A source may generate a packet at any time, but may transmit only at the times specified by the algorithm being used. The generation times are assumed independent and form a Poisson point process with (global) arrival rate λ, measured in packets per slot. When two or more sources transmit so that their messages overlap it is assumed that all the messages are lost and require retransmission. We

further assume that if only one source transmits, the message is received perfectly. This assumption is realistic as long as the probability of collision is much greater than the probability of message error, which will be true if the signal-to-noise ratio is high or forward error correction is used. Finally, we assume that all the users are able to listen to the channel and determine instantaneously whether 0,1 or more than one message were just sent. Alternatively, this information can be provided by a feedback channel.

We are concerned with maximizing the throughput of the channel. We define the capacity, λ^*, of the conflict resolution algorithm to be the supremum over all arrival rates λ such that the expected number of messages generated but not yet successfully transmitted remains bounded. If λ is less than the capacity, then the throughput is λ and the system is stable. If λ exceeds the capacity, then the throughput is at most λ^*.

1.3 Access Strategies

In this section, we discuss a number of schemes for accessing the broadcast channel: Aloha and its modifications, Capetanakis' tree algorithm, and a first-come first-served algorithm proposed by Gallager and modified by Humblet.

1.3.1 Aloha

Our model is a fairly accurate representation of the Aloha system, a radio packet computer communication network developed at the University of Hawaii. In this system, when

a conflict occurs, contending sources wait a random length of time before retransmitting. This scheme has been quoted as having a maximum throughput of 1/2e under the (false) assumption that the total traffic is Poisson (1). A modification to this scheme, called slotted Aloha, has been proposed in which users are constrained to start transmission at discrete times, at one slot intervals (2). This increases the "throughput" to 1/e. Both schemes, however, can be shown to be unstable when the number of sources is infinite (3), although this has not been a problem in practice, due to the small number of actual users.

Another scheme, known as reservation Aloha, has been proposed in which two virtual channels are used (4). A user requests channel capacity over the reservation channel and dynamic TDMA is used to divide the other channel among those who have requested it. This is only useful when the sources send messages that are much longer than the request packets, and since the request channel is still accessed randomly, it can suffer from poor stability and low throughput.

Another Aloha scheme, carrier sense multiple access (5), requires users to "listen" to the channel before transmitting to see if there is a transmission already in progress. It is only useful when the sum of the propagation time between users and the time necessary to detect that the channel is idle is much smaller than a slot. This means that the users

must be relatively close together. It also has stability problems, since many users may attempt to transmit simultaneously when a packet terminates.

1.3.2 The Tree Algorithm

In 1977, Capetanakis introduced a tree algorithm for resolving conflicts (6). In this scheme the sources are each assigned an address which specifies their location on a binary tree. When a conflict occurs, the tree is divided into two halves and the sources in one half attempt retransmission first. The sources in the second half must await retransmission until all conflicts in the first half are resolved. If further conflicts occur upon division, the procedure is repeated. The period of time between the slot in which a conflict first occurs until it is finally resolved is referred to as an epoch. Any new messages arriving during this resolution process must wait until the end of the epoch before attempting transmission.

He then generalized his algorithm to allow the root-node of the tree to be of degree greater than two, choosing it to minimize the expected epoch length. In addition to having good delay characteristics, this scheme is stable for λ less than .430.

1.3.3 Gallager's Algorithm

Gallager (7) has since modified this algorithm to one which is first-come first-served (FCFS), maintains stability and good delay, and has a capacity of .48711. By first-come

first-served, we mean that no message is ever successfully transmitted before another message with an earlier generation time. In Gallager's algorithm, each source in the system maintains a record of the values of three variables: the time at which the source generates a packet to be transmitted, and two system parameters, a lag, d, and an interval length, τ. These parameters represent the state of the system and are constantly being updated. At the start of each new slot, only those sources will send which generated messages during the time interval of length τ (measured in slots) which began d slots past. This interval will be referred to as the transmission interval, and if a conflict occurs in it, it is called a conflict interval.

The operation of this scheme and the updating of d and τ are most easily shown by an example. In figure 1, the arrows along the time axes indicate times at which sources have generated messages for transmission. The heavy vertical bars at the right of each time line show the location of the present slot, and the number between the lines indicates whether the channel sees no message, one message or a conflict. We assume in (1.1) that we begin with some known values of τ and d, and we observe how they are changed. Since only one message was generated in the interval of length τ, beginning d slots ago, there is no conflict in the present slot. Hence the lag parameter is updated according to:

$$d_{new} = d_{old} + 1 - \tau_{old} .$$

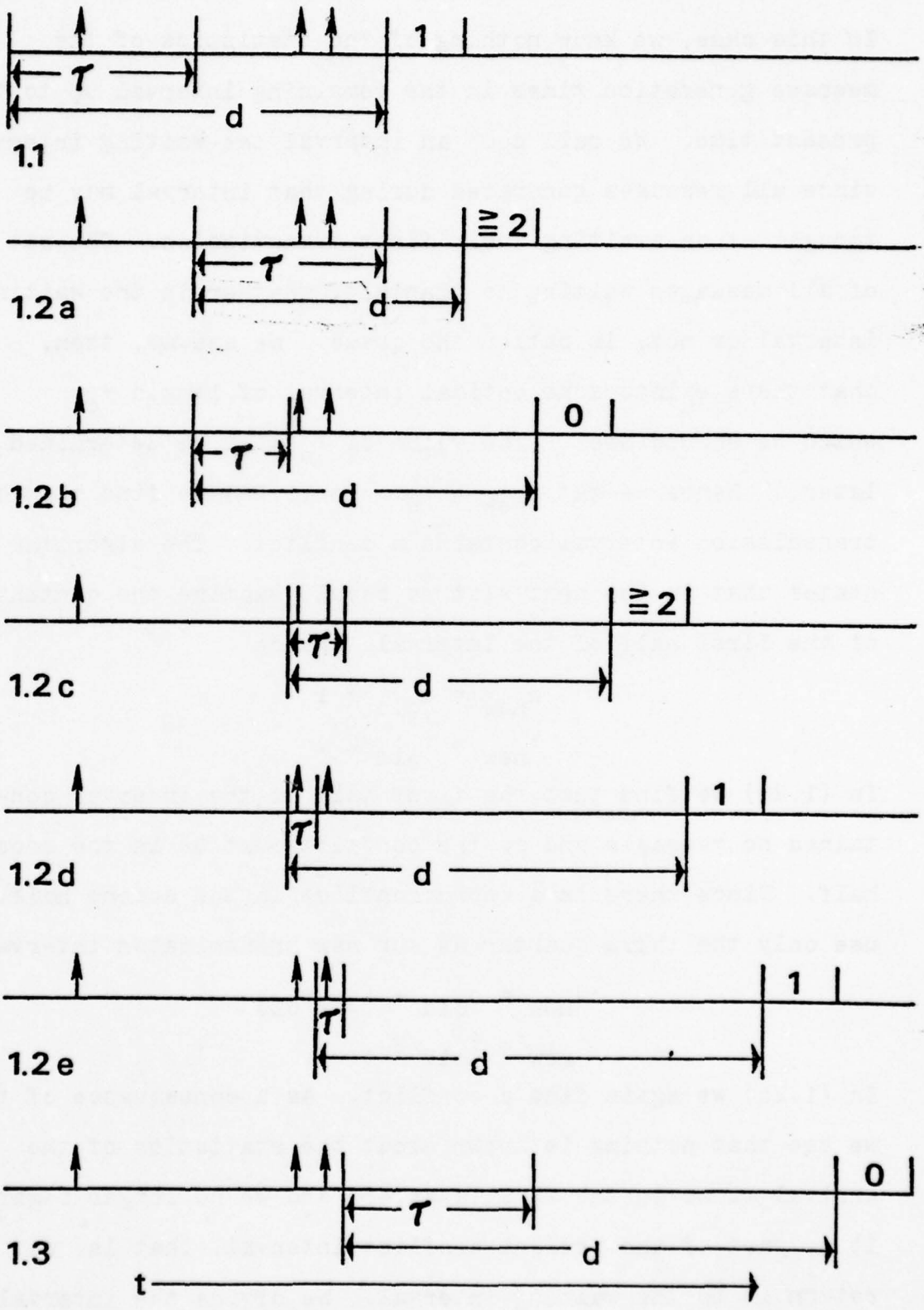


Figure 1

In this case, we know nothing of the statistics of the message generation times in the remaining interval up to the present time. We call such an interval the waiting interval, since all messages generated during that interval may be thought of as awaiting their first transmission. The set of all messages waiting to transmit, whether in the waiting interval or not, is called the queue. We assume, then, that there exists some optimal interval of length τ_o, which we should use. (The value of τ_o will be determined later.) Hence we set $\tau_{new} = \tau_o$. In (1.2a) we find that the transmission interval contains a conflict. The algorithm states that in the next slot we should examine the contents of the first half of the interval. Hence

$$d_{new} = d_{old} + 1$$
$$\tau_{new} = \tau_{old}/2 .$$

In (1.2b) we find that the first half of the interval contained no messages and so the conflict must be in the second half. Since there is a known conflict in the second half, we use only the third quarter as our new transmission interval:

$$d_{new} = d_{old} + 1 - \tau_{old}$$
$$\tau_{new} = \tau_{old}/2 .$$

In (1.2c) we again find a conflict. As a consequence of this we see that nothing is known about the statistics of the arrival times in the final quarter, and we no longer regard it as part of the present conflict interval. That is, we return it to the waiting interval. We divide the interval

again and let:

$$d_{new} = d_{old} + 1 - \tau_{old}$$
$$\tau_{new} = \tau_{old} .$$

In (1.2e) we find only one message sent and there are no further conflicts. The epoch is over and we begin again in (1.3) with:

$$d_{new} = d_{old} + 1 - \tau_{old}$$
$$\tau_{new} = \tau_{o} .$$

Gallager shows that for the system to be stable, λ must be less than or equal to .48711 and that τ_o satisfies $\lambda\tau_o = 1.266$.

At the beginning of a new epoch it is possible that the waiting interval is smaller than τ_o. In this case we may either transmit the arrivals in the waiting interval or wait until we can take an interval of length τ_o. The first choice will probably yield a shorter average delay.

Gallager's algorithm is much like Capetanakis' in that it divides the remaining contending sources in half at the start of each new slot. There are several differences, however, which improve capacity and possibly delay. First, in the case where the first half of a conflict interval contains no messages, so that there is a known conflict in the second half, the tree algorithm sends the messages generated in the whole second half, while Gallager's scheme send only the messages in the third quarter. Next, when a conflict is found in the first half of a conflict interval, Gallager's

scheme returns the second half to the waiting interval, while in an analogous situation, the Capetanakis algorithm still considers the second half of the tree part of the conflict. These differences are probably the major source of improvement. Finally, Gallager's scheme begins a new epoch with an interval whose length has been optimally determined, whereas the Capetanakis algorithm is constrained to choose an integer for the degree of the new optimum tree's root node.

1.3.4 Humblet's Algorithm

Humblet (8) considered Gallager's algorithm and proposed the following modifications. First, the optimal division of a conflict interval is not always in equal parts. Instead it is a function of the length of the conflict interval. Second, consider an interval known to contain at least one message, as in the case when the first part of a conflict interval is found to contain exactly one message. Humblet has shown that if the length of such an interval exceeds a certain threshold, then it is better to use some fraction of that interval, rather than the whole, as the new transmission interval. The values of these optimal divisions for varying interval lengths were solved for by computer and the algorithm using these values was found to have capacity .48775, only marginally greater than before. The reason for this is that for small interval lengths the optimal divisions are very nearly the same as in Gallager's algorithm. In addition, it was found that following this

procedure would never result in an interval containing at least one message whose length exceeds the threshold mentioned above. Hence, after the first division of a conflict interval, the algorithms are almost identical.

1.4 The General First-Come First-Served Algorithm

In this section we describe the general first-come first-served algorithm. We then examine a specific form of it which is an obvious extension of Gallager's and Humblet's algorithms and which is amenable to analysis.

For an algorithm to be FCFS, it must satisfy one of two properties. It must never allow a message to be transmitted when other messages with earlier generation times must wait. Or, if it does, the probability of successful transmission must be zero.

Suppose that we are using a FCFS algorithm to resolve conflicts and that all messages with generation times prior to some t have been successfully transmitted. The algorithm chooses some subset of the queue to transmit in the next slot. We call this the transmission set. Then the most general form of a transmission set which satisfies the first condition will clearly be the set of all messages generated in an interval of the form $[t, t+\tau]$, where τ is some positive real number.

Suppose we have a subset of the queue, S_1, which is known to contain at least one message, but it is not known whether it contains more. Consider the transmission set

which is the union of S_1 and some subset of the queue, S_2. (An example of such a transmission set will be given later in this section, in fig. 2.2.) If the subset of S_2 which is disjoint from S_1 is not empty, then a conflict occurs and the second condition holds. If the subset of S_2 which is disjoint from S_1 is empty, then S_1 must be the set of messages generated in an interval of the form $[t,t+\tau]$ for the first condition to hold. Since we assume that the algorithm never chooses a transmission set that is known to contain a conflict, we cannot know in advance if the subset of S_2 disjoint from S_1 is non-empty. Hence, S_1 must be the set of messages generated in an interval of the form $[t,t+\tau]$ to insure that one of the two conditions hold.

The general first-come first-served algorithm must use transmission sets which are one of the two types described above. In this thesis we restrict our attention to the case where transmission sets consist only of the arrivals in intervals of the form $[t,t+\tau]$. This is done for ease of analysis and because it seems likely that there is no advantage in using the more general sets. We call this the single interval algorithm (SIA).

Let us consider the algorithm in more detail. Suppose we begin with an interval of length s known to contain at least one message. Our next transmission interval will be of length α. For some values of s we expect the optimal α to be less than or equal to s (fig. 2.1a), but in some cases α is greater

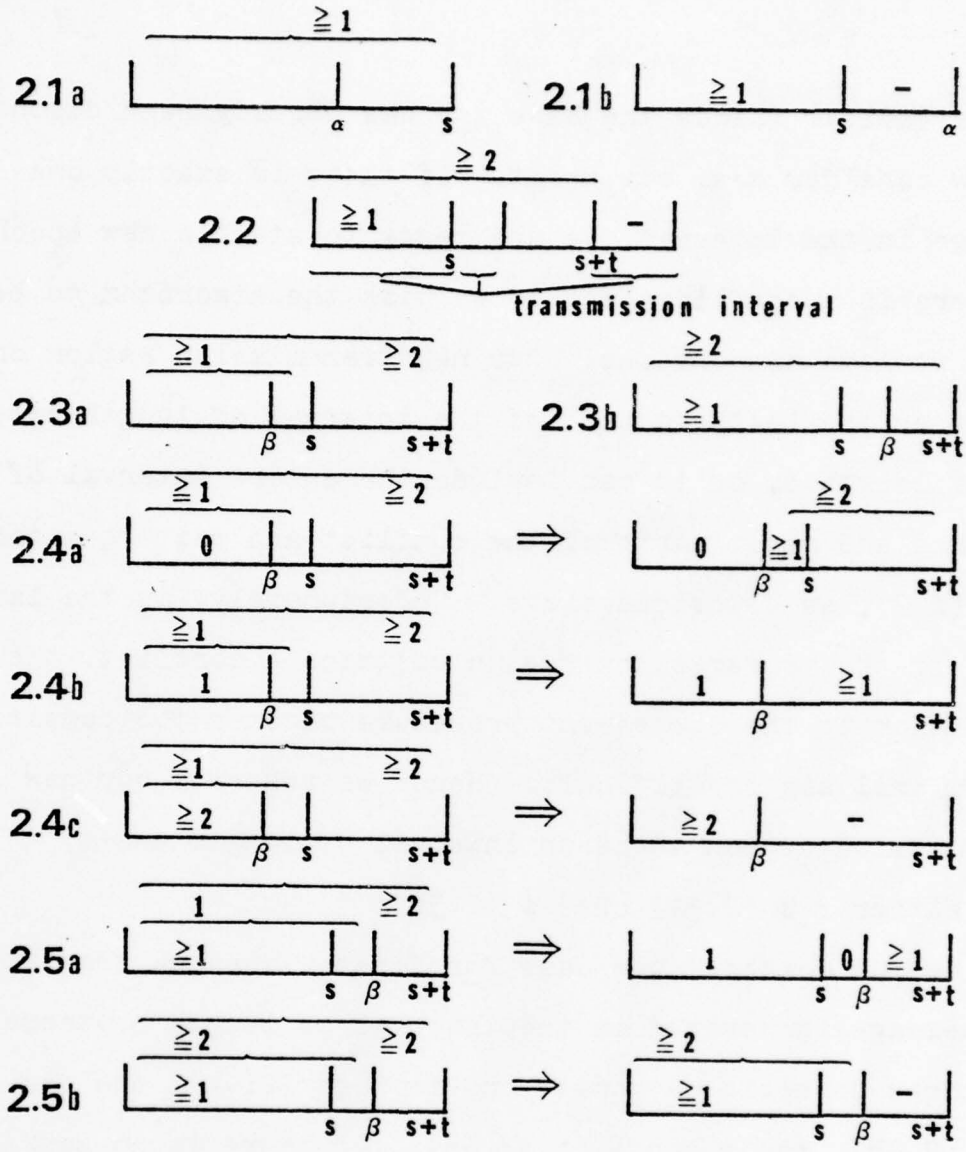


Figure 2

The numbers above the intervals indicate the number of messages a segment is known to contain. A dash indicates that nothing is known about the statistics of that segment.

than s (2.1b). Since the case $\alpha \leq s$ has already been discussed, we now consider $\alpha > s$, say $\alpha = s+t$. If there is exactly one message in the interval, we are ready to start a new epoch. If there is a conflict, and if we wish the algorithm to be FCFS, we have two choices. Our new transmission region can be either the leftmost part of the interval of length s+t, say of length β, or it can include the entire interval of length s and other parts of the conflict and waiting intervals (2.2), as described above. Unfortunately, in the latter case, if the transmission region contains a conflict, the definition of the subsequent procedure becomes too complex and its analysis is difficult. Hence we restrict our new transmission region to be an interval of length $\beta < s+t$, with either $\beta \leq s$ (2.3a) or $\beta > s$ (2.3b).

Let us consider the case $\beta \leq s$ first. When we transmit the messages generated in the interval of length β, there are three possible outcomes: no message (2.4a), one message (2.4b), and a conflict (2.4c). If there is no message, then we have left an interval of length $s-\beta$ containing at least one message and a combined interval of length $s+t-\beta$ containing a conflict. We process this interval in the same fashion that we did the original interval, except that a different β is chosen.

In the case where one message is found, we are left with an interval of length $s+t-\beta$ which contains at least one message, with the statistics derived below.

Throughout the rest of the thesis we use a shorthand notation, where "k in s" is taken to mean "there are k messages in an interval of length s." Because the generation statistics are Poisson, the numbers of messages in disjoint intervals are independent. Therefore:

$$\Pr(k \text{ in } s+t-\beta \mid 1 \text{ in } \beta,\ \geq 1 \text{ in } s,\ \geq 2 \text{ in } s+t)$$

$$= \frac{\Pr(k \text{ in } s+t-\beta,\ 1 \text{ in } \beta,\ \geq 1 \text{ in } s,\ \geq 2 \text{ in } s+t)}{\Pr(1 \text{ in } \beta,\ \geq 1 \text{ in } s,\ \geq 2 \text{ in } s+t)}$$

$$= \frac{\Pr(1 \text{ in } \beta)\Pr(k \text{ in } s+t-\beta,\ \geq 0 \text{ in } s-\beta,\ \geq 1 \text{ in } s+t-\beta)}{\Pr(1 \text{ in } \beta)\Pr(0 \text{ in } s-\beta,\ \geq 1 \text{ in } s+t-\beta)}$$

$$= \frac{\Pr(k \text{ in } s+t-\beta,\ \geq 1 \text{ in } s+t-\beta)}{\Pr(1 \text{ in } s+t-\beta)}$$

$$= \Pr(k \text{ in } s+t-\beta \mid \geq 1 \text{ in } s+t-\beta).$$

This result is crucial to the implementation and analysis of the algorithm, since it allows us to consider the processing of only two types of intervals: one which contains at least one message, and one which contains a conflict, and has at least one message in its first part. After we have found that the interval of length $s+t-\beta$ contains at least one message, we process it in the appropriate manner, either dividing it or augmenting it.

Finally, if there is a conflict in the interval of length β, we treat it as a conflict interval and return the remainder to the waiting interval.

In the case where $\beta > s$, there are two possible outcomes: one successful transmission (2.5a) or a conflict (2.5b).

If we find one message in the interval, then we are left with an interval of length s+t-β which contains at least one message and we treat it as described earlier. When there is a conflict, we have an interval of length s containing at least one message and a combined interval of length β with a conflict. We return the remaining interval of length s+t-β to the waiting interval, and divide the interval of length β as described before.

1.5 Thesis Outline

The thesis is organized into three chapters and an appendix. Chapter 1 is this introduction. The analysis is carried out in Chapters 2 and 3 and the computer programs used in the analysis are given in the appendix. Here we consider the objectives and results of these chapters.

In Chapter 2, we model the single interval contention resolution algorithm as a Markov process with a continuous state space. We define functions representing the expected times required to transmit all of the messages generated in a particular interval of time. Three such functions are required, corresponding to the three classes of states of the Markov system. We define cost functions for the process in terms of the drift, the difference between the lag at the time when the system next enters S_0, a renewal state, and the lag at the present time. We see that the equations for the expected cost per epoch are identical to those developed for minimizing the time required to process an

interval. A technique for solving these equations for Humblet's algorithm is described, and it is explained why this method is unsuitable for analysis of the single interval algorithm.

In the third chapter, we describe some of the basic results of Markovian decision theory: the value iteration algorithm and the Odoni bound. We generalize these results for continuous state spaces. We describe how the value functions may be approximated by calculating the values for a discrete subset of the state space. As a result of computer analysis of these functions, we see that for all practical purposes, Humblet's algorithm is the best steady state single interval algorithm in terms of capacity. Errors in the analysis due to the discretization of the state space are discussed. The insensitivity of the results to the number of points used to approximate the state space, leads us to conclude that such errors are negligible. Suggestions for further research follow.

In the appendix, some notes on the computer programs are given, as well as the program listings.

II. Analysis of the Algorithm by Expected Drift

In this chapter we model the algorithm as a problem in Markovian decision theory with a cost structure based on the difference in the lag before and after state transitions. We describe how this method is used to analyze Humblet's algorithm, and why it is unsuitable for the analysis of the single interval algorithm.

2.1 The Algorithm as a Markov Process

In this section we model the single interval algorithm as a Markov process. The state of the system at any time is determined by the length and statistics of the previous transmission interval. In section 1.4 we saw that there are three types of intervals we will need to consider, corresponding to three classes of states for the system. The system is in state S_0 if the last transmission ended an epoch. The system is in state $S_1(s)$ if, as a result of the last transmission, we know only that an interval of length s contains at least one message. The system is in state $S_2(s,t)$ if we know that an interval of length s contains at least one message, and a longer interval of length $s+t$ contains a conflict. Note that the moments at which the system returns to state S_0 are renewal points. When the system is in state S_0, the algorithm chooses an interval of length τ as the next transmission interval. In state $S_1(s)$ a transmission interval of length $\alpha(s)$ is

chosen. In state $S_2(s,t)$ we choose an interval of length $\beta(s,t)$. Figure 3 shows a state transition diagram for the algorithm. The number by an arrow connecting two states indicates the number of messages that must be found in the transmission interval for that state transition to occur.

While this Markov process has a continuous state space, and therefore an uncountably infinite number of states, it is worth noting that all but a countably infinite number of these states are transient. If the system is in some state other than S_0 at time t=0, then because we assume the algorithm is stable, the system must eventually enter S_0. From S_0, under a stationary policy, there are only a countably infinite number of states the system may enter. This can be seen by observing that state transitions occur at discrete times, and that from a given state there are never more than a finite number of transitions the system may make. Hence, all states that cannot be reached from S_0 are transient.

Having modeled our algorithm as a Markov process, we would like to use this model to determine the values of $\tau(\lambda)$, $\alpha(s,\lambda)$ and $\beta(s,t,\lambda)$ that will maximize throughput. (Here we write τ,α and β as functions of λ to emphasize their dependence on λ.) One way of doing this would be to assign a cost or reward structure to the state transitions and apply the methods of Markovian decision theory to determine the optimal controls τ,α and β. In this chapter we will consider a cost structure where we seek to minimize the difference

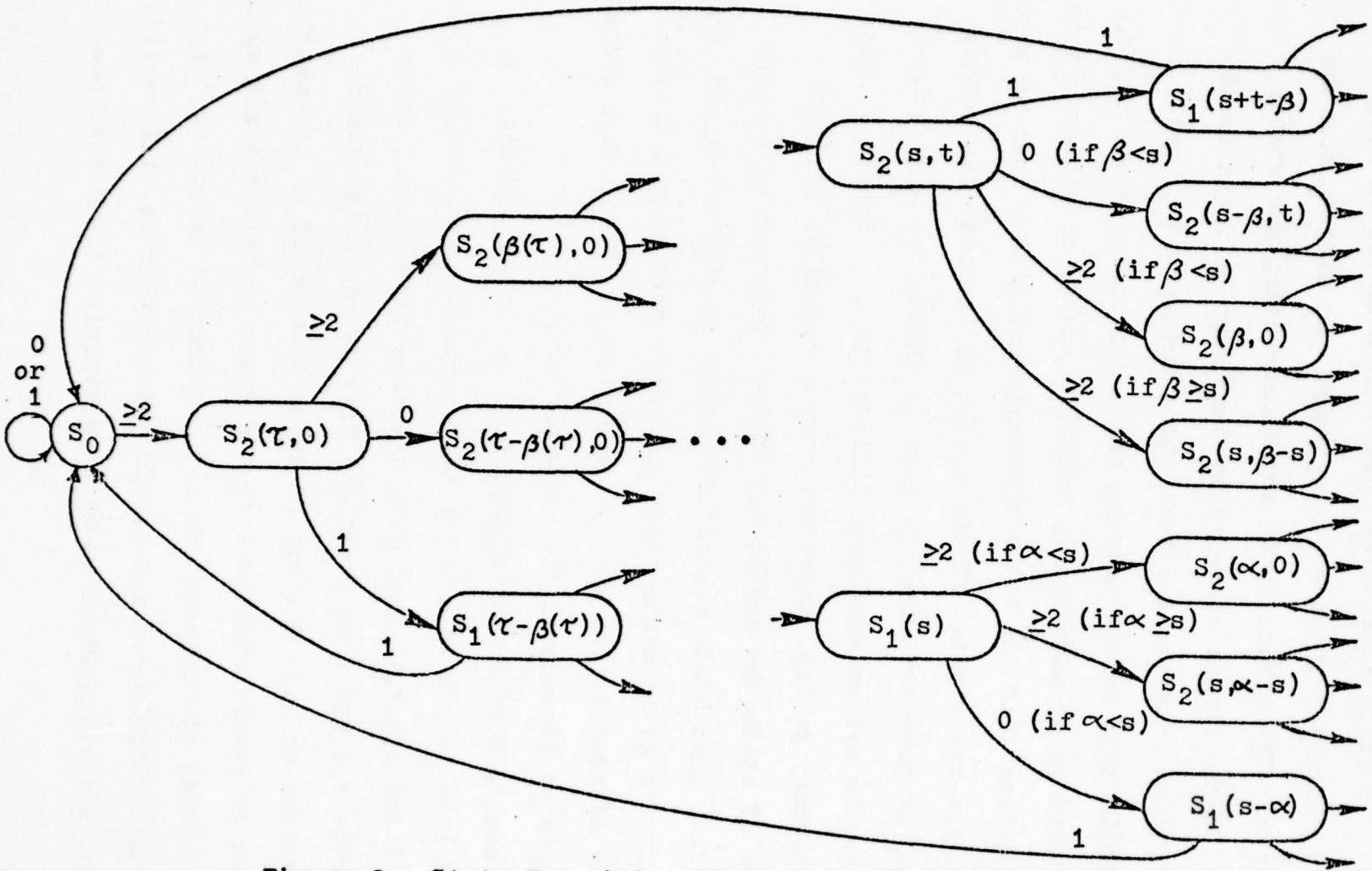


Figure 3. State Transition Diagram for Markov Process Model of Single Interval Algorithm

in the lag between successive epochs. In Chapter 3, we consider a reward structure.

2.2 Expected Times to Resolve Intervals

In this section we develop a set of equations whose solution gives the optimal values of $\tau(\lambda)$, $\alpha(s,\lambda)$ and $\beta(s,t,\lambda)$, and the capacity, λ^*. Our approach is to minimize the expected time required to resolve conflicts while maintaining finite delay. To aid in the discussion we introduce the following terminology. We say that an interval has been processed when all the messages generated in that interval have been successfully transmitted. We say that an interval has been resolved when the system next enters the state S_0.

We begin by defining three functions. Let $M_0(x)$ be the expected time to process an interval of length x/λ containing messages generated by a Poisson process. Let $M_1(x,y)$ be the expected time to process an interval of length $(x+y)/\lambda$, where there is at least one message in the segment of length x/λ and the arrivals in the segment of length y/λ are Poisson. Let $M_2(x,y,z)$ be the expected time to process an interval of length $(x+y+z)/\lambda$, where there is at least one message in the segment of length x/λ, at least two in the segment of length $(x+y)/\lambda$, and the arrivals in the segment of length z/λ are Poisson. The variables x,y and z may be thought of as the lengths of the intervals in a normalized time frame where the Poisson arrival rate is 1 message per time unit. It is important to note that though the arguments of these

functions are given in normalized time, the values of the functions are in real time. If T, A and B are the normalized lengths of the transmission intervals used when the system is in the states S_0, S_1 and S_2 respectively, and if T is fixed and A and B are chosen optimally for that T, then from these definitions and from the discussion of section 1.4 we have:

$$(2.1a)\quad M_0^T(x) = \begin{cases} 1 + \Pr(\geq 2 \text{ in } x)M_2^T(x,0,0) + \Pr(0 \text{ or } 1 \text{ in } T)M_0^T(0) & x<T \\ 1 + \Pr(\geq 2 \text{ in } T)M_2^T(T,0,x-T)+\Pr(0 \text{ or } 1 \text{ in } T)M_0^T(x-T) & x\geq T \end{cases}$$

$$(2.1b)\quad M_1^T(x,y)=\min_A \begin{cases} 1+\Pr(0 \text{ in } A|\geq 1 \text{ in } x)M_1^T(x-A,y)+ \\ \quad \Pr(1 \text{ in } A|\geq 1 \text{ in } x)M_0^T(x+y-A)+ \\ \quad \Pr(\geq 2 \text{ in } A|\geq 1 \text{ in } x)M_2^T(A,0,x+y-A) & A\leq x \\ 1+\Pr(1 \text{ in } A|\geq 1 \text{ in } x)M_0^T(x+y-A)+ \\ \quad \Pr(\geq 2 \text{ in } A|\geq 1 \text{ in } x)M_2^T(x,A-x,x+y-A) & A>x \end{cases}$$

$$(2.1c)\quad M_2^T(x,y,z)=\min_B \begin{cases} 1+\Pr(0 \text{ in } B|\geq 1 \text{ in } x,\geq 2 \text{ in } x+y)M_2^T(x-B,y,z)+ \\ \quad \Pr(1 \text{ in } B|\geq 1 \text{ in } x,\geq 2 \text{ in } x+y)M_1^T(x+y-B,z)+ \\ \quad \Pr(\geq 2 \text{ in } B|\geq 1 \text{ in } x,\geq 2 \text{ in } x+y)M_2^T(B,0,x+y+z-B) & B\leq x \\ 1+\Pr(1 \text{ in } B|\geq 1 \text{ in } x,\geq 2 \text{ in } x+y)M_1^T(x+y-B,z)+ \\ \quad \Pr(\geq 2 \text{ in } B|\geq 1 \text{ in } x,\geq 2 \text{ in } x+y)M_2^T(x,A-x,x+y+z-A) & \\ & x<B< x+y \end{cases}$$

Now we define three new functions, F_0, F_1 and F_2, in terms of M_0, M_1 and M_2. These functions are also dependent on T, but we do not write the T for simplicity. Let:

(2.2a) $M_0^T(x) = F_0(x)$

(2.2b) $M_1^T(x,y) = F_1(x) + F_0(y)$

(2.2c) $M_2^T(x,y,z) = F_2(x,y) + F_0(z)$

Combining eqs. 2.1 and 2.2 we get:

(2.3a) $$F_0(x) = \begin{cases} 1+F_0(0)+\Pr(\geq 2 \text{ in } x)F_2(x,0) & x \leq T \\ 1+F_0(x-T)+\Pr(\geq 2 \text{ in } T)F_2(T,0) & x > T \end{cases}$$

(2.3b) $$F_1(x)+F_0(y) = \min_A \begin{cases} 1+\Pr(0 \text{ in } A \mid \geq 1 \text{ in } x)(F_1(x-A)+F_0(y))+ \\ \quad \Pr(1 \text{ in } A \mid \geq 1 \text{ in } x)F_0(x+y-A)+ \\ \quad \Pr(\geq 2 \text{ in } A \mid \geq 1 \text{ in } x)(F_2(A,0)+F_0(x+y-A)) & A \leq x \\ 1+\Pr(1 \text{ in } A \mid \geq 1 \text{ in } x)F_0(x+y-A)+ \\ \quad \Pr(\geq 2 \text{ in } A \mid \geq 1 \text{ in } x)(F_2(x,A-x)+F_0(x+y-A)) & A > x \end{cases}$$

(2.3c) $$F_2(x,y)+F_0(z) = \min_B \begin{cases} 1+\Pr(0 \text{ in } B \mid \geq 1 \text{ in } x, \geq 2 \text{ in } x+y)(F_2(x-B,y)+F_0(z))+ \\ \quad \Pr(1 \text{ in } B \mid \geq 1 \text{ in } x, \geq 2 \text{ in } x+y)(F_1(x+y-B)+F_0(z))+ \\ \quad \Pr(\geq 2 \text{ in } B \mid \geq 1 \text{ in } x, \geq 2 \text{ in } x+y)(F_2(B,0)+F_0(x+y+z-B) & B \leq x \\ 1+\Pr(1 \text{ in } B \mid \geq 1 \text{ in } x, \geq 2 \text{ in } x+y)(F_1(x+y-B)+F_0(z))+ \\ \quad \Pr(\geq 2 \text{ in } B \mid \geq 1 \text{ in } x, \geq 2 \text{ in } x+y)(F_2(x,B-x)+F_0(x+y+z-B)) & x < B < x+y \end{cases}$$

Since, for $x > T$, $F_0(x)-F_0(x-T) = 1 + \Pr(\geq 2 \text{ in } T)F_2(T,0) =$ a constant, $F_0(x)$ is asymptotically linear in x. Hence, for large x, we may write $F_0(x) = kx/\lambda + b$. Let $k^* = k/\lambda$. Then the system will be stable so long as k^* is less than $1/\lambda$.

We may use this expression for F_0 to rewrite eq. (2.3a):

$$k^*x = 1 + k^*(x-T) + \Pr(\geq 2 \text{ in } T)F_2(T,0) \quad \text{or}$$

$$(2.4) \quad k^* = \frac{1 + \Pr(\geq 2 \text{ in } T)F_2(T,0)}{T} \quad 1/\lambda \quad .$$

We may also rewrite eq. (2.3b):

$$F_1(x)+k^*y+b=\min_A \begin{cases} 1+b+\Pr(0 \text{ in } A|\geq 1 \text{ in } x)(F_1(x-A)+k^*y)+ \\ \quad \Pr(1 \text{ in } A|\geq 1 \text{ in } x)(k^*y+k^*(x-A))+ \\ \quad \Pr(\geq 2 \text{ in } A|\geq 1 \text{ in } x)(F_2(A,0)+k^*y+k^*(x-A)) \quad A\leq x \\ 1+b+\Pr(1 \text{ in } A|\geq 1 \text{ in } x)(k^*y+k^*(x-A))+ \\ \quad \Pr(\geq 2 \text{ in } A|\geq 1 \text{ in } x)(F_2(x,A-x)+k^*y+k^*(x-A)) \quad A>x \end{cases}$$

or

$$(2.5a) \quad F_1(x)=\min_A \begin{cases} 1+\Pr(0 \text{ in } A|\geq 1 \text{ in } x)F_1(x-A)+\Pr(\geq 2 \text{ in } A|\geq 1 \text{ in } x)F_2(A,0)+ \\ \quad (1-\Pr(0 \text{ in } A|\geq 1 \text{ in } x))k^*(x-A) \quad A\leq x \\ 1+k^*(x-A)+\Pr(\geq 2 \text{ in } A|\geq 1 \text{ in } x)F_2(x,A-x) \quad A>x \end{cases} .$$

Similarly, we may rewrite eq. (2.3c):

$$(2.5b) \quad F_2(x,y)=\min_B \begin{cases} 1+\Pr(0 \text{ in } B|\geq 1 \text{ in } x,\geq 2 \text{ in } x+y)F_2(x-B,y)+ \\ \quad \Pr(1 \text{ in } B|\geq 1 \text{ in } x,\geq 2 \text{ in } x+y)F_1(x+y-B)+ \\ \quad \Pr(\geq 2 \text{ in } B|\geq 1 \text{ in } x,\geq 2 \text{ in } x+y)(F_2(B,0)+k^*(x+y-B)) \quad B\leq x \\ 1+\Pr(1 \text{ in } B|\geq 1 \text{ in } x,\geq 2 \text{ in } x+y)F_1(x+y-B)+ \\ \quad \Pr(\geq 2 \text{ in } B|\geq 1 \text{ in } x,\geq 2 \text{ in } x+y)(F_2(x,B-x)+k^*(x+y-B)) \end{cases}$$

$$x<B< x+y.$$

Hence we wish to find the smallest value of k^* such that when $A(x)$ and $B(x,y)$ are chosen to minimize $F_1(x)$ and $F_2(x,y)$, there exists some value of T for which eq. (2.4) is satisfied. Then the capacity will be $\lambda^*=1/k^*$.

2.3 Minimizing Expected Drifts

In addition to their definition in terms of expected times to resolve intervals, the functions $F_0(x)$, $F_1(x)$ and $F_2(x,y)$ may also be interpreted in terms of the expected drift, which we define below.

We define a random variable $U(S)$, called the drift. If the system is in state S at time t, then $U(S)$ is the difference between the lag at the time when the system next enters the state S_0 and the lag at time t. Let $U_0(S_0)$, $U_1(x)=U(S_1(x/\lambda))$ and $U_2(x,y)=U(S_2(x/\lambda, y/\lambda))$. U_0 is the drift in lag between one epoch and the preceding epoch. If the system is to be stable, then it is clear that $E(U_0)<0$, otherwise the delay will become infinite as the number of epochs goes to infinity. The capacity of the algorithm will be the largest value of λ for which a policy exists that has $E(U_0)$ less than zero.

Next we show that $F_1(x)=E(U_1(x)) + x/\lambda$ and $F_2(x,y)=E(U_2(x,y)) + (x+y)/\lambda$. Therefore, minimizing F_1 and F_2 for a fixed λ is equivalent to minimizing the drift. To see this, consider Figure 4. At time t the system is in state S with a transmission interval of length 1 and lag d. At time t' the system has returned to S_0 with lag d'. The

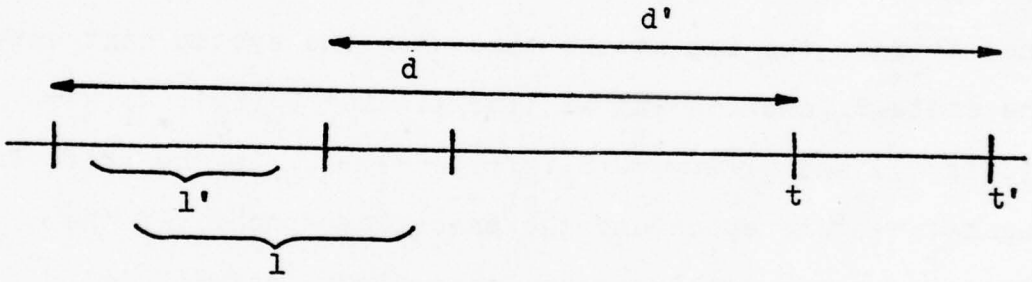


Figure 4

length of the processed interval is l'. Then we have

$$U(S) = d' - d = (t' - t) - l'$$

and

$$E(U(S)) + l = E(t' - t) + E(l - l')$$

$$= E(\text{time to return to } S_0) +$$

$$E(\text{time returned to the waiting interval}) -$$

$$E(\text{time taken from the waiting interval}) .$$

Examining eq. (2.5) we see that F_1 and F_2 may also be defined as the expected time to resolve an interval, plus the time returned to the waiting interval and minus the time taken from the waiting interval.

With this interpretation for $F_2(x,y)$, it is easy to see that:

$$E(U_0) = 1 + Pr(\geq 2 \text{ in } T)F_2(T,0) - T/\lambda .$$

Hence the condition that $E(U_0)$ be less than zero is equivalent to the condition of eq. (2.4).

2.4 Solving the Equations for U_0, $F_1(x)$ and $F_2(x,y)$

In this section we discuss methods for solving eqs. (2.4) and (2.5). In their present form these equations are very difficult to work with. To simplify them, we define two new functions:

$$g_1(x) = Pr(\geq 1 \text{ in } x)F_1(x)$$

$$g_2(x,y) = Pr(\geq 1 \text{ in } x, \geq 2 \text{ in } x+y)F_2(x,y) .$$

Substituting these equations into eq. (2.4), the stability condition becomes:

$$1 + g_2(T,0) - T/\lambda = 0 \quad . \tag{2.6}$$

From eq. (2.5) we get:

$$(2.7a)\quad g_1(x)=\min_A \begin{cases} (1-e^{-x}) + e^{-A}g_1(x-A) + g_2(A,0) + (1-e^{-A})\dfrac{(x-A)}{\lambda^*} & A\leq x \\ (1-e^{-x})\left(1-\dfrac{(A-x)}{\lambda^*}\right) + g_2(x,A-x) & A>x \end{cases}$$

$$(2.7b)\quad g_2(x,y)=\min_B \begin{cases} (1-e^{-x}(1+xe^{-y})) + e^{-B}g_2(x-B,y) + g_2(B,0) + \\ \quad Be^{-B}g_1(x+y-B) + (1-(1+B)e^{-B})\dfrac{(x+y-B)}{\lambda^*} & B\leq x \\ (1-e^{-x}(1+xe^{-y})) + xe^{-B}g_1(x+y-B) + g_2(x,B-x) + \\ \quad (1-e^{-x}-xe^{-B})\dfrac{(x+y-B)}{\lambda^*} & x<B<x+y \end{cases} .$$

First we consider the problem of solving these equations for the case where we restrict $A\leq x$. (This is the method used by Humblet to analyze his algorithm.) Then the second argument of g_2 will always be 0 and we also have $B<x$. Thus we see that $g_1(x)$ and $g_2(x,0)$ are defined in terms of g_1 and g_2 at smaller arguments. In addition, it is easily verified that $g_1(0)=0$ and $g_2(0,0)=0$. Thus we may construct the functions g_1 and g_2 by starting with some very small x and solving simultaneously for $g_1(x)$ and $g_2(x,0)$, linearly interpolating between $g_1(x)$ and $g_1(0)$, and $g_2(x,0)$ and $g_2(0,0)$, where needed. We then proceed by incrementing x and solving at each step by interpolating between the previously calculated values. This will lead us into trouble, however, since it can be shown that for

small values of x, $g_2(x,0)$ is quadratic in x, not linear. To circumvent this difficulty, we define a new function, $g_2'(x,y)=g_2(x,y)/x$. The changes to eq. (2.7) are obvious and we do not present them here.

The following argument shows that $g_1(x)$ is linear in small x and that $g_2(x,0)$ is quadratic in small x. As x goes to 0,

$$\Pr(k \text{ in } x \mid \geq 1 \text{ in } x) = \begin{cases} 1 & \text{if } k=1 \\ 0 & \text{if } k\neq 1 \end{cases} .$$

Therefore, the expected length of time required to process this interval is just one slot; that is, $F_1(0)=1$. So for small x, $g_1(x) \approx 1-e^{-x} \approx x$.

Also, as x goes to 0,

$$\Pr(k \text{ in } x \mid \geq 2 \text{ in } x) = \begin{cases} 1 & \text{if } k=2 \\ 0 & \text{if } k\neq 2 \end{cases} .$$

For small x, the optimal B is approximately x/2. Then,

$$\Pr(0 \text{ in } x/2 \mid \geq 2 \text{ in } x) = \Pr(\geq 2 \text{ in } x/2 \mid \geq 2 \text{ in } x) = \tfrac{1}{4}$$

and

$$\Pr(1 \text{ in } x/2 \mid \geq 2 \text{ in } x) = \tfrac{1}{2} .$$

From eq. (2.5b) we get

$$F_2(0,0) = 1 + \tfrac{1}{2}F_2(0,0) + \tfrac{1}{2}F_1(0)$$

and therefore

$$F_2(0,0) = 3 .$$

So for small x, $g_2(x,0) \approx 3(1-(1+x)e^{-x}) \approx 3(1-(1-x)(1-x+x^2/2)) \approx (3/2)x^2$.

Humblet has written a computer program to construct the

funtions $g_1(x)$ and $g_2(x,0)$ by this technique. After the functions are calculated, the drift is found from g_2. Different values of λ are used to determine the highest value that gives a non-positive drift, $E(U_0)$. The optimal initial interval length, T, is found to be approximately 1.27 and the throughput is approximately .488.

This method has several problems. One problem is that the program cannot solve for λ^* directly. It must be run for many different values of λ to find the greatest one. Also, solving for more than two decimal places of accuracy in the optimal initial interval length requires an excessive amount of computation, especially in view of that fact that the program will be run many times to find the capacity. Finally, because the program relies on finding λ^* such that the drift is zero, the result is very sensitive to small errors in g_1 and g_2.

Attempts were made to solve for the capacity by this method when A is not restricted to be less than or equal to x. However, because $g_1(x)$ and $g_2(x,y)$ are defined in terms of g_1 and g_2 at both smaller and greater arguments, it was not possible to construct the functions for successively larger x and y by interpolating between previously calculated values.

III. Analysis of the Algorithms by Markovian Decision Theory with a Reward Structure

In this chapter we analyze the algorithms by use of a reward structure, where a reward of one is given for every successful transmission. We want to maximize the expected reward per state transition, which is called the gain. Clearly the gain is equal to the throughput and the maximum gain is the capacity. Before proceeding with this analysis, we review some basic results of Markovian decision theory which were derived for Markov processes with finite state spaces, and show how they can be generalized to the continuous state space model.

3.1 Markovian Decision Theory and Continuous State Spaces

In this section we introduce two basic methods of Markovian decision theory, the value iteration algorithm and the Odoni bound, and generalize them for continuous state spaces.

3.1.1 The Value Iteration Algorithm

The object of Markovian decision theory is to find optimal policies for controlling a Markov process in order to maximize the gain. One method of determining such a policy is through use of the value iteration equations (9). The following model is used. We have a finite state space, with states i=1,. . .,N, and for each state i, a finite choice of controls k=1,. . . ,n_i. We have a set of state

transition probabilities, p_{ij}^{k}, which are dependent on the control k used, and a set of state transition rewards, r_{ij}^{k}, which may or may not be dependent on the control. The following quantities are defined:

$$(3.1) \qquad q_i^k = \sum_{j=1}^{N} p_{ij}^k \; r_{ij}^k$$

$$(3.2) \qquad v_i^{k(n+1)}(n+1) = q_{ij}^k + \sum_{j=1}^{N} p_{ij}^{k(n+1)} \; v_j^{k(n)}(n) \; .$$

Equation (3.1) gives the expected reward on the next transition if the system is in state i and control k is applied. Equation (3.2) defines the value functions, where $v_i^{k(n+1)}(n+1)$ is the total expected reward over the next n+1 transitions, if the system begins in state i and the control k(n) is applied when there n transition remaining. The values of $v_i(0)$ are assumed to be given and are called terminal rewards.

If the values of k_i are chosen at each n such that

$$(3.3) \qquad v_i(n+1) = \max_k q_{ij}^k + \sum_{j=1}^{N} p_{ij}^k \; v_j(n)$$

then the sequence of vectors k(n) represents an optimal time-dependent policy. This is the optimality principle. If the values of k are held constant for all n, we say that the policy is stationary. Schweitzer (10) has shown that, for any set of initial values $v_i(0)$, if all stationary

policies result in single-chain aperiodic Markov processes, then as $n\to\infty$, $v_i(n)\to ng+v_i$, where g is the gain and the v_i satisfy :

$$(3.4) \qquad v_i = \max_k \; q_i^k - g + \sum_{j=1}^{N} p_{ij}^k \, v_j \quad .$$

In addition, any vector k which satisfies this equation defines a policy with the maximum gain.

Schweitzer also introduces a notation for considering a continuous state space with finite volume. Careful examination of his proof of the above indicates that a similar theorem for continuous spaces could be shown in an analogous manner using the general notation, and adding some simple assumptions about continuity. We make the following conjecture.

We consider a system with a state space S, which is the union of a finite number of sets with finite volume. That is,

$$S=\bigcup_{i=1}^{N} S_i \quad \text{where } S_i \varepsilon R^{n_i} \quad \text{and } \int_{S_i} dx < \infty \quad .$$

The state of the system is represented by a vector $x_i \varepsilon S_i$, $1 \le i \le N$. (Note that if the sets S_i are disjoint, the subscript of x may be omitted.) For each state there is a set of controls $K(x_i)$. From each state $x_i \varepsilon S_i$, for each control $k \varepsilon K(x_i)$, for each $j=1,\ldots,N$, there exists at most one point $y_j \varepsilon S_j$ which the system may enter. The control k which is chosen determines a set of probabilities $P_{ij}^k(x)$,

where $P_{ij}^{k}(x)$ is the probability that the system will go from state $x \epsilon S_i$ to state $y \epsilon S_j$ if the control k is applied. The reward functions, $r_{ij}^{k}(x)$ are defined in the obvious manner. With this notation, we rewrite eqs. (3.1), (3.3) and (3.4).

$$(3.5) \qquad q_i^{k}(x) = \sum_{j=1}^{N} P_{ij}^{k}(x)\, r_{ij}^{k}(x)$$

$$(3.6) \qquad v_i(x,n+1) = \sup_{k} \; q_i^{k(x)}(x) + \sum_{j=1}^{N} P_{ij}^{k(x)}(x)\, v_j(y_j,n)$$

$$(3.7) \qquad v_i(x) = \sup_{k} \; q_i^{k}(x) + \sum_{j=1}^{N} P_{ij}^{k}(x)\, v_j^{k}(y_j)$$

The sup in eqs. (3.6) and (3.7) can be replaced by max if we assume either that the control sets are finite, or if they are compact and the right hand sides of the equations are continuous in k for all points in the state space.

We conjecture that, for any initial set of values $v_i(x,0)$, if all stationary policies result in single chain aperiodic Markov processes, then as $n \to \infty$, $v_i(x,n) \to ng + v_i(x)$, where g is the gain and the v_i satisfy eq. (3.7).

While we do not attempt to prove this conjecture here, it will be seen later that when we calculate approximations to the $v_i(x,n)$ for our problem, these value functions do indeed converge.

3.1.2 The Odoni Bound

Odoni (11) has provided upper and lower bounds for the gain that can be calculated at each iteration of the value iteration algorithm. We can use these bounds to give an estimate of the gain and upper bound the error of our estimate. These bounds generalize easily to the continuous state space. For the continuous state space model described in the previous section, we give the following theorem.

Theorem. Let $q_i^k(x)$ and $v_i(x,n)$ be defined as in eqs. (3.5) and (3.6). Let $v_i(x,0)$ and $r_{ij}(x)$ be bounded for all $1 \leq i \leq N$, $x_i \in S_i$. Then,

i) $L''(n) \equiv \sup_{i,x} (v_i(x,n+1)-v_i(x,n))$ is monotone decreasing in n.

ii) $L'(n) \equiv \inf_{i,x} (v_i(x,n+1)-v_i(x,n))$ is monotone increasing in n.

iii) $L'(n) \leq \liminf_{n'\to\infty} \frac{v_i(x,n')}{n'} \leq \limsup_{n'\to\infty} \frac{v_i(x,n')}{n'} \leq L''(n)$ for all n.

Proof. The proof of i) and ii) is completely analogous to the proof given by Odoni, and we do not repeat it here. To prove iii) observe that for fixed n, for any $n' > n$,

$$\begin{aligned} v_i(x,n') &= (v_i(x,n')-v_i(x,n'-1)) + \cdots \\ &\quad + (v_i(x,n+1)-v_i(x,n)) + v_i(x,n) \\ &\leq \sup(v_i(x,n')-v_i(x,n'-1)) + \cdots \\ &\quad + \sup(v_i(x,n+1)-v_i(x,n)) - v_i(x,n) \\ &= L''(n'-1) + \cdots + L''(n) + v_i(x,n) \\ &\leq (n'-n)L''(n) + \sup v_i(x,n) \quad , \end{aligned}$$

where the last inequality holds because L''(n) is monotone decreasing in n. Therefore,

$$\frac{v_i(x,n')}{n'} \leq (1-\frac{n}{n'})\, L''(n) + \frac{\sup v_i(x,n)}{n'} \quad .$$

Now $\sup v_i(x,n) \leq \sup v_i(x,0) + n \sup r_{ij}^{k}(x)$ is bounded. Taking the limit as $n' \to \infty$ we have:

$$\lim_{n'\to\infty} \sup \frac{v_i(x,n')}{n'} \leq L''(n) \quad .$$

By a similar argument, we have:

$$L'(n) \leq \lim_{n'\to\infty} \inf \frac{v_i(x,n')}{n'}$$

and we are done.

This theorem gives upper and lower bounds on the maximum gain, even when the iterated value functions do not converge. We use this theorem in the computer programs to bound the capacity of Gallager's and Humblet's algorithms, and the single interval algorithm.

3.2 Discretizing the State Space to Approximate the Value Functions

With the notation developed in section 3.1, we are ready to write value functions for our model of the single interval contention resolution algorithm. Our state space S is the union of three sets, $S_0=\{1\}$, $S_1=\{x \mid x>0\}$ and $S_2=\{(x,y) \mid x>0,\ y\geq 0\}$. The control sets for S_0 and S_1 are the same: $K=\{T \mid T>0\}$ and the control set for the state $S_2(x,y)$ is $\{T \mid 0<T<x+y\}$. The state space that we use in

this chapter differs from that used in Chapter 2, in that the variables x and y here represent the normalized lengths of intervals, that is, the units in which x are measured are chosen such that the arrival rate of the messages is 1. The controls T, A and B are normalized lengths also.

The reward, which is independent of the control, is 1 for a successful transmission and 0 otherwise. Hence the expected reward on the next transition is just the probability of finding one message in the transmission interval. From eqs. (3.5) and (3.6) we have:

$$(3.8a)\quad v_{S_0}(n+1) = \max_{T>0} \; Te^{-T} + (1+T)e^{-T} v_{S_0}(n) + (1-(1+T)e^{-T}) v_{S_2(T,0)}(n)$$

$$(3.8b)\quad v_{S_1(x)}(n+1) = \max_{A>0} \begin{cases} \dfrac{e^{-A}-e^{-x}}{1-e^{-x}} v_{S_1(x-A)}(n) + \dfrac{Ae^{-A}}{1-e^{-x}}(1+v_{S_0}(n)) \\ \quad + \dfrac{1-(1+A)e^{-A}}{1-e^{-x}} v_{S_2(A,0)}(n) \quad A<x \\ \dfrac{xe^{-A}}{1-e^{-x}}(1+v_{S_0}(n)) \\ \quad + \dfrac{1-e^{-x}-xe^{-A}}{1-e^{-x}} v_{S_2(x,A-x)}(n) \quad A \geq x \end{cases}$$

$$(3.8c)\quad v_{S_2(x,y)}(n+1) = \max_{0 \leq B < y} \begin{cases} \dfrac{e^{-B}-e^{-x}-(x-B)e^{-(x+y)}}{1-e^{-x}-xe^{-(x+y)}} v_{S_2(x-B,y)}(n) \\ \quad + \dfrac{B(e^{-B}-e^{-(x+y)})}{1-e^{-x}-xe^{-(x+y)}}(1-v_{S_1(x+y-B)}(n)) \end{cases}$$

$$\begin{cases} + \dfrac{1-(1+B)e^{-B}}{1-e^{-x}-xe^{-(x+y)}} \, v_{S_2(B,0)}(n) & B<x \\ \dfrac{x(e^{-B}-e^{-(x+y)})}{1-e^{-x}-xe^{-(x+y)}}(1+v_{S_1(x+y-B)}(n)) & \\ + \dfrac{1-e^{-x}-xe^{-B}}{1-e^{-x}-xe^{-(x+y)}} \, v_{S_2(x,B-x)}(n) & x\leq B<x+y \end{cases}$$

We use initial values $v_{S_0}(0)=v_{S_1(x)}(0)=v_{S_2(x,y)}(0)=0$ for all $x>0$, $y\geq 0$. With this choice of terminal rewards, we may write closed form expressions for $v_{S_0}(1)$ and $v_{S_1(x)}(1)$, but not for $v_{S_2(x,y)}(1)$ and not for any of the value functions when $n>1$. Therefore we have written a computer program to solve for the value functions.

The program can calculate the value functions only for a finite number of points. Hence we approximate the state space with a discrete grid. Since we know that at each iteration there is some finite optimal initial interval length, $T(n)$, we may limit the control set for S_0 to the set $\{T \mid 0<T\leq T'<\infty\}$, where T' is chosen such $T(n)\leq T'$ for all n. It should be clear that for the state $S_1(x)$ we will never want to choose a transmission interval longer than T', and we may also limit the control set for S_1. Noting that the control set for $S_2(x,y)$ is already bounded, we see that the sets of non-transient states are bounded, since the system never has to consider an interval longer than T'. Hence our discrete approximation to the state space has a finite number of points.

When writing the computer program, it was necessary to guess in advance approximately where the bounds on the set of non-transient states lie. Since we can only store a small number of points, and since we would like the grid to be as fine as possible for accuracy, it is important to choose the bounds on the grid well. The choice of these bounds is discussed in the Appendix.

Since we can only calculate the functions at a finite number of points, and since this calculation will often require values of the functions from the previous iteration at points which are not stored in the grid, it is necessary to approximate the values at these points. The program does this by linearly interpolating between the values of the two points on the grid which are closest to the point whose value we are estimating. This method introduces some error into the calculation of the value functions. It is difficult to analyze this error, but the results of the computer program indicate that is quite small for even very coarse grids. We discuss this in more detail in section 3.5.

This method of estimating the values of points not on the grid creates a problem when we calculate the value functions at $S_1(\Delta x)$ or $S_2(\Delta x,y)$, where Δx is the smallest x on the grid. Calculating $v_{S_1(\Delta x)}(n-1)$ and $v_{S_2(\Delta x,y)}(n-1)$ both require knowledge of $v_{S_1(x)}(n)$ and $v_{S_2(x,y)}(n)$ for values of $x<\Delta x$. But there are no points on the grid below

$v_{S_1(\Delta x)}(n)$ and $v_{S_2(\Delta x,y)}(n)$ to use in the interpolation.

Hence, we define $v_{S_1(0)}(n) = \lim_{x\to 0} v_{S_1(x)}(n)$ and $v_{S_2(0,y)}(n) = \lim_{x\to 0} v_{S_2(x,y)}(n)$ for $y\neq 0$. We let $v_{S_1(0)}(0) = v_{S_2(0,y)}(0) = 0$.

From eqs. (3.8b) and (3.8c) we get:

$$(3.8d)\quad v_{S_1(0)}(n+1) = \max_{A\geq 0} \; e^{-A}(1+v_{S_0}(n)) + (1-e^{-A})v_{S_2(0,A)}(n)$$

$$(3.8e)\quad v_{S_2(0,y)}(n+1) = \max_{B\geq 0} \; \frac{e^{-B}-e^{-y}}{1-e^{-y}}\,(1+v_{S_1(y-B)}(n)) + \frac{1-e^{-B}}{1-e^{-y}}\, v_{S_2(0,B)}(n) \quad .$$

Defining $v_{S_2(0,0)}(n)$ presents a problem, since the value function is discontinuous at this point. That is,

$$\lim_{x\to 0} v_{S_2(x,0)}(n) \neq \lim_{y\to 0} v_{S_2(0,y)}(n) \quad .$$

As x goes to 0, the state $S_2(x,0)$ represents the knowledge that a very small interval contains two messages. As seen in section 2.4, it will require an average of 3 slots to process these messages. But as y goes to 0, the state $S_2(0,y)$ represents the knowledge that two very small intervals each contain one message, requiring only 2 slots to process. Hence, the discontinuity of the value functions arises. To overcome this problem, the program is written to interpolate between the values $\lim_{x\to 0} v_{S_2(x,0)}(n)$ and $v_{S_2(\Delta x,0)}(n)$ whenever a value for $v_{S_2(x,0)}(n)$ is needed where $0<x<\Delta x$. Similarly, the program interpolates between $\lim_{y\to 0} v_{S_2(0,y)}(n)$ and $v_{S_2(0,\Delta y)}(n)$ whenever a value for

$v_{S_2(0,y)}(n)$ is needed, where $0<y<\Delta y$ and Δy is the smallest y on the grid. These limits are given by:

$$(3.8f)\quad \lim_{x\to 0} v_{S_2(x,0)}(n+1) = \tfrac{1}{2}(1 + v_{S_1(0)}(n) + \lim_{x\to 0} v_{S_2(x,0)}(n))$$

$$(3.8g)\quad \lim_{y\to 0} v_{S_2(0,y)}(n+1) = 1 + v_{S_1(0)}(n) \quad .$$

In addition to discretizing the state space, we must also discretize the control sets in order to find the maxima in eqs. (3.8(a-e)). Since the control sets have already been shown to be bounded, discretizing insures that they have a finite number of elements.

3.3 The Results of the Computer Programs

In this section we give the results of the computer programs used to analyze the value functions and bound the throughput of the continuous interval algorithm. We first discuss the optimal time-dependent policy for finite horizon and then the optimal stationary policy. Listings of the programs and some descriptive notes are given in the Appendix.

3.3.1 The Optimal Time Dependent Policy

The program results show that for a finite horizon, for the state $S_1(x)$, the optimal control A is greater than x for x less than a threshold dependent on n, which we call $x_T(n)$. The control function A(x,n) is discontinuous as $x_T(n)$. For x less than $x_T(n)$, A is increasing in x and decreasing

in n. The threshold is also decreasing in n. This is illustrated in Figure 5, which shows plots of A(x) for n=3,4 and 5. It was not possible to plot A(x) for n greater than 5, since $x_T(n)$ for larger n is less than the smallest positive value of x for which $v_{S_1(x)}(n)$ is calculated. However, for all successive n, the program shows A(0)>0, with A(0) converging to .057.

We conjecture that for all finite n, there exists some positive x such that for x' less than x, A(x',n)>x'. This need not be true as n goes to infinity, even though A(0)>0 for all n, since the control function is discontinuous.

The discontinuity of A(x) stems from the fact that the function $v_{S_1(x)}{}^{A}(n)$, when considered as a function of A for fixed x and n, has two local maxima. (See Figure 6.) For example, $v_{S_1(.06)}{}^{A}(3)$ has local maxima of 1.7455 at A=.06 and 1.7492 at A=.3, and hence the optimal control is greater than x. The threshold for n=3 occurs at x=.07, where the local maxima are both 1.7428 at A=.07 and A=.3. For x=.08, the maxima are 1.7402 at A=.08 and 1.7366 at A=.32. Hence we see a discontinuity in the control function at x=.07, but not in the value function.

For the states $S_1(x)$, there is a second threshold, x_T', such that for $x_T<x<x_T'$, A(x)=x, but for $x \geq x_T'$, A(x)<x. For large n, x_T'= .78 for both Humblet's algorithm and the SIA. The slope of A(x) is discontinuous at this point. It was this fact that originally led us to suspect that

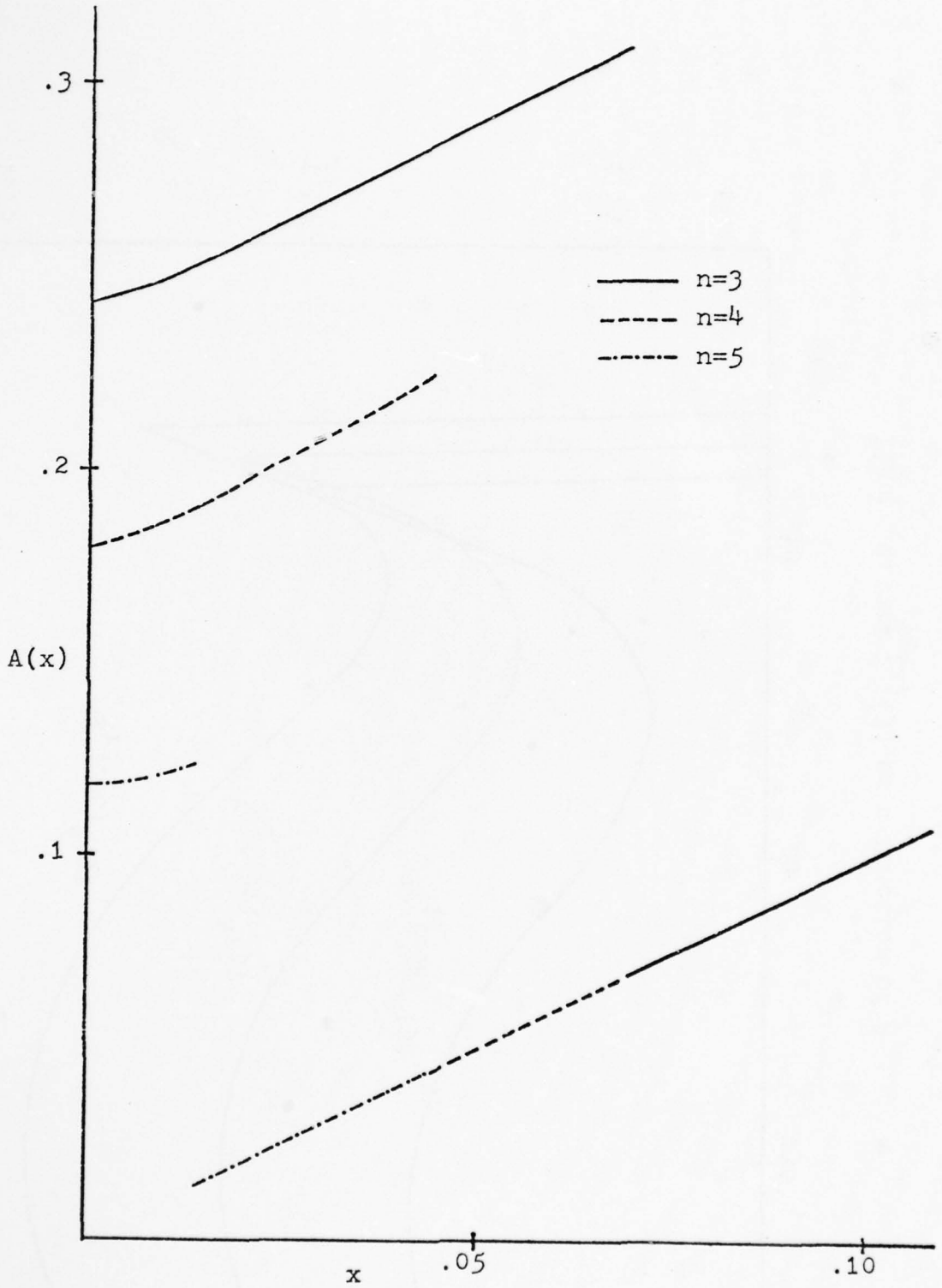


Figure 5. A(x) versus x

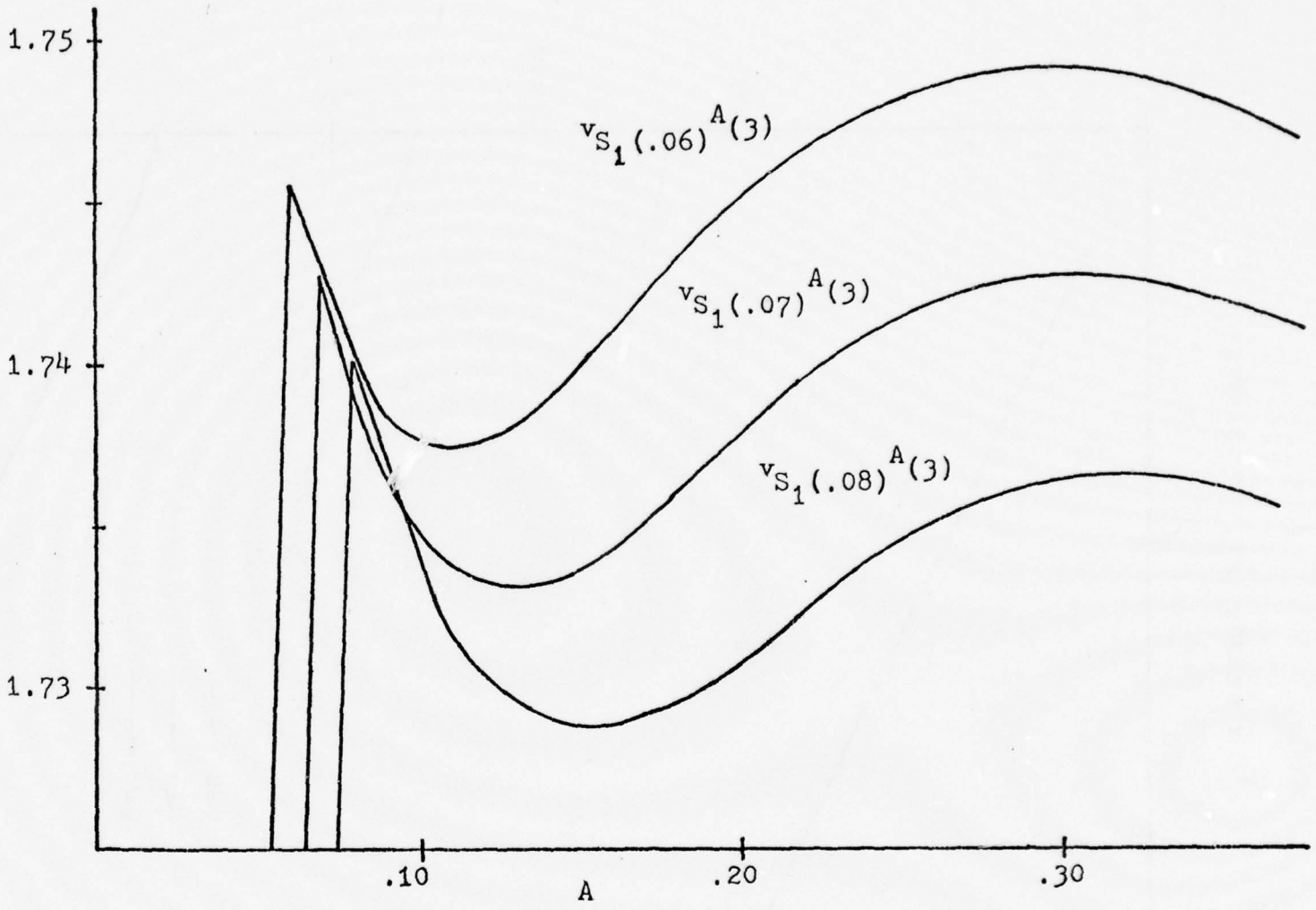


Figure 6. $v_{S_1(x)}(3)$ as a function of A

we might want $A(x)>x$ for $x<.78$. We expected that letting A exceed x for $x<.78$ would remove this discontinuity in $A(x)$.

The controls for the states $S_2(x,0)$ are also the same as for Humblet's algorithm. In addition, it was found that $B(x,y,n)$ was always less than or equal to x, for $y<.96$.

3.3.2 The Optimal Stationary Policy

The program written to calculate the value functions was designed to stop when the difference between the upper and lower bounds for the gain became less than 10^{-5}. This occurred after 18 iterations. We assume that the policy at this point is the optimal stationary policy. This policy for the single interval algorithm is indistinguishable from the policy for Humblet's algorithm. It is not clear, however, that the algorithms are identical. As mentioned in the preceding section, for $n>5$, $x_T(n)$ is too small for the program to calculate. This threshold decreases so rapidly, that we conjecture that $x_T(n)\to 0$ as $n\to\infty$. Even if this is not true, the value x_T to which $x_T(n)$ converges is so small, that the probability that the system will ever enter a state $S_1(x)$, where $x\leq x_T$ is nearly zero. Thus, for all practical purposes, Humblet's algorithm is the optimal steady state single interval algorithm.

As mentioned earlier, using Humblet's algorithm, the system will never encounter a state $S_1(x)$ where $x\geq .78$. Therefore, $A=x$ for all non-transient states $S_1(x)$. This fact, and a table of values for $B(x)$ completely specify

the algorithm. A table of B(x) is included on the next page.

3.4 Error in the Value Functions due to Linear Approximations

In this section, we explain why we believe that the errors introduced into the value functions by linear interpolation are negligible

If the function being approximated is very nearly linear over the region between the points where it is being interpolated, then the error in the interpolation will be small. Even at the first iteration, the functions $v_{S_1(x)}(1)$ and $v_{S_2(x,0)}(1)$ are very nearly linear. After the program converges to a stationary policy, $v_{S_1(x)}(n)$ is even closer to being linear and $v_{S_2(x,y)}(n)$ is nearly linear in both x and y.

If the function is monotonic between the points where it is being interpolated, the error will be no greater than the difference between the values of the function at those two points. This difference may be made as small as desired by choosing the grid fine enough with respect to the slope of the function. Since the slope of the functions is low, we can use a relatively coarse grid and still keep errors small.

A powerful argument for neglecting possible interpolation errors is that program results are almost completely independent of the size of the grid used. In particular, a program was written to analyze Gallager's algorithm

X	B	X	B	X	B
0.01	0.005	0.44	0.214	0.87	0.410
0.02	0.010	0.45	0.218	0.88	0.415
0.03	0.015	0.46	0.223	0.89	0.419
0.04	0.020	0.47	0.228	0.90	0.423
0.05	0.025	0.48	0.232	0.91	0.428
0.06	0.030	0.49	0.237	0.92	0.432
0.07	0.035	0.50	0.242	0.93	0.437
0.08	0.040	0.51	0.246	0.94	0.441
0.09	0.045	0.52	0.251	0.95	0.445
0.10	0.050	0.53	0.256	0.96	0.450
0.11	0.055	0.54	0.260	0.97	0.454
0.12	0.060	0.55	0.265	0.98	0.458
0.13	0.064	0.56	0.270	0.99	0.463
0.14	0.069	0.57	0.274	1.00	0.467
0.15	0.074	0.58	0.279	1.01	0.471
0.16	0.079	0.59	0.284	1.02	0.476
0.17	0.084	0.60	0.288	1.03	0.480
0.18	0.089	0.61	0.293	1.04	0.484
0.19	0.094	0.62	0.297	1.05	0.489
0.20	0.099	0.63	0.302	1.06	0.493
0.21	0.104	0.64	0.307	1.07	0.497
0.22	0.108	0.65	0.311	1.08	0.501
0.23	0.113	0.66	0.316	1.09	0.506
0.24	0.118	0.67	0.320	1.10	0.510
0.25	0.123	0.68	0.325	1.11	0.514
0.26	0.128	0.69	0.329	1.12	0.519
0.27	0.133	0.70	0.334	1.13	0.523
0.28	0.137	0.71	0.338	1.14	0.527
0.29	0.142	0.72	0.343	1.15	0.531
0.30	0.147	0.73	0.347	1.16	0.536
0.31	0.152	0.74	0.352	1.17	0.540
0.32	0.157	0.75	0.357	1.18	0.544
0.33	0.161	0.76	0.361	1.19	0.548
0.34	0.166	0.77	0.365	1.20	0.552
0.35	0.171	0.78	0.370	1.21	0.557
0.36	0.176	0.79	0.374	1.22	0.561
0.37	0.181	0.80	0.379	1.23	0.565
0.38	0.185	0.81	0.383	1.24	0.569
0.39	0.190	0.82	0.388	1.25	0.573
0.40	0.195	0.83	0.392	1.26	0.578
0.41	0.199	0.84	0.397	1.27	0.582
0.42	0.204	0.85	0.401	1.28	0.586
0.43	0.209	0.86	0.406	1.29	0.590

Table 1

Values of B as a function of x for Humblet's Algorithm

approximating $v_{S_1(x)}(n)$ at 100 points between 0 and 1.0, and $v_{S_2(x,0)}(n)$ at 150 points between 0 and 1.5. After 18 iterations the capacity is bounded below by .48711 and above by .48712, and the optimal initial interval length is $1.266/\lambda$. These are exactly the values found by Gallager using other methods. Another program was written approximating $v_{S_1(x)}(n)$ and $v_{S_2(x,0)}(n)$ by only two points each: $v_{S_1(0)}(n)$ and $v_{S_1(.65)}(n)$, $v_{S_2(0,0)}(n)$ and $v_{S_2(1.3,0)}(n)$. After 18 iterations, this program bounded the capacity by .48662 below and .48663 above, and found an optimal initial interval of length $1.271/\lambda$. These results represent errors of less than one-tenth and four-tenths of one percent, respectively.

A program analyzing Humblet's algorithm using the first grid described above gave a capacity between .48775 and .48776 with an optimal initial interval of $1.275/\lambda$. The program which analyzed the single interval algorithm with a large coarse grid (described in the Appendix) converges to an equivalent stationary policy which has the same capacity and initial interval length, even though the grid uses only 1/6 as many points.

This insensitivity of the method to grid size and density, and the good agreement with results obtained by other methods indicates that our approximations to the value functions are quite accurate.

3.5 Suggestions for Further Research

Many fundamental questions about the capacity of contention resolution algorithms remain unanswered. Is the best single interval algorithm the best first-come first-served? Is the best first-come first-served the best possible algorithm? How may these algorithms be modified for the case where not all users know the outcome of each trial? How may they be modified when the users are trying to access more than one facility?

A. APPENDIX

Program Notes

In this appendix we include listings of the two Fortran programs that were used to calculate the value function and to bound the capacity.

The first program calculates approximations to $v_{S_1(x)}(n)$ and $v_{S_2(x,y)}(n)$ using a large, coarse grid. The grid size was determined from the results of Humblet's program, described in section 2.4. For his algorithm it was found that, in the $S_1(x)$ state, for values of x up to .78, the optimal policy chooses A=x. For larger x, A will be less than x. For this reason, we thought it unlikely that the single interval algorithm would choose A>x for x>.78. For safety, however, we chose the grid to allow A>x for x up to .96. We expected (incorrectly) that for small x, A would be much larger than x, and as x approached .78, A would decrease to x. Thus we chose to calculate $v_{S_2(x,y)}(n)$ only for x and y such that $x+y \leq .96$, $y>0$. Since Humblet's algorithm found an optimal initial interval length larger than Gallager's, it seemed likely that the SIA would use an even larger T. Hence, we calculate $v_{S_2(x,0)}(n)$ for x up to 1.47.

The program was initially written for Δx (the distance between grid points) equal to .06. At each iteration the program would calculate value functions for approximately 200 states. Changing a single card in the program (line 4)

would change Δx from .06 to .01. This was never done, since it would increase the number of states from 200 to 5000, and the program would have been too costly to run. The program results, however, indicated that the bounds of the initial grid were much larger than required. The program was re-written with a small, fine grid using approximately the same number of states as the first grid.

The second program uses $\Delta x=.01$. Since the first program showed that for $x \geq .12$, $A \leq x$, we restricted the grid to allow $A>x$ for x up to .15. This extra margin was included in case the threshold for A might change due to the finer discretization. Values for $v_{S_1(x)}(n)$ were calculated only for $x \leq .79$, since it had been shown that states $S_1(x)$ for larger x were transient. Since the first program showed that the largest $A>x$ ever used is .3 when x=.06, we chose to calculate $v_{S_2(x,y)}(n)$ for $0 \leq x \leq .15$ and $0 \leq y \leq .3$. Since the largest T ever used is 1.28, we calculated $v_{S_2(x,0)}(n)$ for x up to 1.29.

The second program also eliminated two costly routines that were found unnecessary. The control B for the state $S_2(x,y)$ was found always less than or equal to x, so the second program searched a much smaller interval for B. For $v_{S_2(0,y)}(n)$, B is positive only for very large y, corresponding to transient states, so we let the second program choose $B(0,y)=0$, without doing a search.

The increment for the controls T, A and B, was chosen

to be $\Delta x/10$. Hence, the finer grid gives more accuracy not only by calculating the functions at more points, but also by allowing more precise determination of the optimal controls.

Even for the coarse grid, however, it is not practical to test all the possible controls for each state. Hence the search for the optimal control is carried out in two parts. First, the control set is searched over a coarse grid, with increments equal to Δx. Then a fine search is made, using increments of $\Delta x/10$, over an interval of width $2\Delta x$, centered at the optimal control found from the coarse search. Maximizing a function in this way can lead to trouble if the function has two peaks and the true maximum lies between the points tested in the coarse search. However, the difference between the value functions at a control T and at $T+\Delta x$ is small enough that any error of this type is negligible.

```
C PROGRAM TO CALCULATE V0,V1(X) AND V2(X,Y) USING A LARGE COARSE GRID.
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION V0(2),V1(2,97),V2(2,51),VV(2,98,97)
      DIMENSION AA(97),BB(51),PTIN(97),PB(97)
      M1=6
      M2=M1+1
      DEL=.01D0
      DEL1=DEL*M1
      DELP=.01D0
      DELP1=DELP*M1
C INITIALIZE CONTENTS OF MATRICES.
      V1(1,1)=-1.D0
      VV(1,98,97)=0.D0
      VV(2,98,97)=0.D0
      I=2
      V0(I)=0.D0
      DO 7 J=1,51
    7 V2(I,J)=0.D0
      DO 8 J=1,97
      PTIN(J)=(J-1)*DEL
      V1(I,J)=0.D0
      DO 8 K=1,97
    8 VV(I,J,K)=0.D0
      DO 1 N=1,25
      I=2-(N-(N/2)*2)
      J=1+N-(N/2)*2
C CALCULATE V0 AND T.
      VMAX=0.D0
      TMAX1=.99D0
C COARSE SEARCH FOR T.
      DO 10 L=1,47
      T=TMAX1+L*DEL
      TEX=DEXP(-T)
      IT=3+L
      VN=T*TEX+(1.D0+T)*TEX*V0(J)+V2(J,IT)*(1.D0-(1.D0+T)*TEX)
      IF (VN.GE.VMAX) TMAX=T
```

```
 10    IF (VN.GE.VMAX) VMAX=VN
C FINE SEARCH FOR T.
      DO 11 L=1,19
      T=TMAX-.01D0+L*DELF
      TEX=DEXP(-T)
      AUX=T/DEL
      IT=AUX
      TDIF=AUX-IT
      TDIF1=1.D0-TDIF
      IT=IT-96
      VN=2*TEX+(1.D0+T)*TEX*V0(J)+(V2(J,IT)*TDIF1+V2(J,IT+1)*TDIF)
     &*(1.D0-(1.D0+T)*TEX)
      IF (VN.GE.VMAX) TMAX1=T
 11   IF (VN.GE.VMAX) VMAX=VN
      V0(I)=VMAX
      T=TMAX1
      X=V0(I)-V0(J)
      XMIN=X
      XMAX=X
C CALCULATE VV(0,Y) AND B(0,Y), B> OR =0.
      DO 29 L=M2,97,M1
      LM=98-L
      Y=(L-1)*DEL
      EY=DEXP(-Y)
      VN=1.D0+V1(J,L)
      B=0.D0
 28   VMAX=VN
      B=B+DELF1
      EB=DEXP(-B)
      AUX=B/DEL1
      IB=AUX
      BDIF=AUX-IB
      BDIF1=1.D0-BDIF
      IBY=L-(IB+1)*M1
      IB=IB*M1+1
      VN=(EB-EY)*(1.D0+V1(J,IBY)*BDIF+V1(J,IBY+M1)*BDIF1)+
```

```
     &(1.D0-EB)*VV(J,1,IB+M1)*BDIF
      IF (IB.EQ.1) VN=(VN+(1.D0-EB)*(1.D0+V1(I,1))*BDIF1)/(1.D0-EY)
      IF (IB.GT.1) VN=(VN+(1.D0-EB)*VV(J,1,IB)*BDIF1)/(1.D0-EY)
      IF (VN.GT.VMAX) GO TO 28
      VV(I,98,LB)=B-DELF1
      VV(I,1,L)=VMAX
      X=VV(I,1,L)-VV(J,1,L)
      IF (X.LT.XMIN) XMIN=X
 29   IF (X.GT.XMAX) XMAX=X
C CALCULATE V1(0) AND A(0).
      VMAX=0.D0
C COARSE SEARCH FOR A.
      DO 15 L=1,97,M1
      A=(L-1)*DEL
      EA=DEXP(-A)
      VN=EA*(1.D0+V0(J))+(1.D0-EA)*VV(J,1,L)
      IF (VN.GE.VMAX) AMAX1=A
 15   IF (VN.GE.VMAX) VMAX=VN
      AMAX=AMAX1
C FINE SEARCH FOR A.
      DO 16 L=1,19
      A=AMAX1-DEL1+L*DELF1
      IF (A.LT.0) GO TO 16
      EA=DEXP(-A)
      AUX=A/DEL1
      IA=AUX
      ADIF=AUX-IA
      ADIF1=1.D0-ADIF
      IA=IA*M1+1
      VN=EA*(1.D0+V0(J))+(1.D0-EA)*VV(J,1,IA+M1)*ADIF
      IF (IA.EQ.1) VN=VN+(1.D0-EA)*(1.D0+V1(I,1))*ADIF1
      IF (IA.GT.1) VN=VN+(1.D0-EA)*VV(J,1,IA)*ADIF1
      IF (VN.GE.VMAX) AMAX=A
      IF (VN.GE.VMAX) VMAX=VN
 16   CONTINUE
      AA(1)=AMAX
```

```
      V1(I,1)=VMAX
      X=V1(I,1)-V1(J,1)
      IF (X.LT.XMIN) XMIN=X
      IF (X.GT.XMAX) XMAX=X
C CALCULATE V1(X) AND A(X).
      DO 20 L=M2,97,M1
      X=(L-1)*DEL
      EX=DEXP(-X)
      VMAX=0.D0
C COARSE SEARCH FOR A.
      DO 21 LA=M2,97,M1
      A=(LA-1)*DEL
      EA=DEXP(-A)
      IF (A.LT.X) VN=(EA-EX)*V1(J,L-LA+1)+A*EA*(1.D0+V0(J))
     &+(1.D0-(1.D0+A)*EA)*VV(J,LA,1)
      IF (A.GE.X) VN=X*EA*(1.D0+V0(J))+(1.D0-EX-X*EA)*VV(J,L,LA-L+1)
      IF (VN.GE.VMAX) AMAX1=A
21    IF (VN.GE.VMAX) VMAX=VN
      AMAX=AMAX1
C FINE SEARCH FOR A.
      DO 24 LA=1,19
      A=AMAX1-DEL1+LA*DELF1
      EA=DEXP(-A)
      IF (A.GE.X) GO TO 22
      AUX=A/DEL1
      IA=AUX
      ADIF=AUX-IA
      ADIF1=1.D0-ADIF
      IAX=L-(IA+1)*M1
      IA=IA*M1+1
      VN=(EA-EX)*(V1(J,IAX)*ADIF+V1(J,IAX+M1)*ADIF1)+A*EA*(1.D0+V0(J))+
     &(1.D0-(1.D0+A)*EA)*(VV(J,IA,1)*ADIF1+VV(J,IA+M1,1)*ADIF)
      GO TO 23
22    AUX=(A-X)/DEL1
      IAX=AUX
      AXDIF=AUX-IAX
```

```
      AXDIF1=1.D0-AXDIF
      IAX=IAX*M1+1
      VN=X*EA*(1.D0+V0(J))+(1.D0-EX-X*EA)*(VV(J,L,IAX)*AXDIF1
     &+VV(J,L,IAX+M1)*AXDIF)
 23   IF (VN.GE.VMAX) AMAX=A
 24   IF (VN.GE.VMAX) VMAX=VN
      AA(L)=AMAX
      V1(1,L)=VMAX/(1.D0-EX)
      X=V1(I,L)-V1(J,L)
      IF (X.LT.XMIN) XMIN=X
 20   IF (X.GT.XMAX) XMAX=X
C CALCULATE VV(0,0).
      VV(I,1,1)=(VV(J,1,1)+V1(J,1)+1.D0)/2.D0
      X=VV(I,1,1)-VV(J,1,1)
      IF (X.LT.XMIN) XMIN=X
      IF (X.GT.XMAX) XMAX=X
C CALCULATE VV(X,Y) AND B(X,Y), X>0.
      DO 30 K=M2,97,M1
      K1=98-K
      KB=99-K
      X=(K-1)*DEL
      EX=DEXP(-X)
      IF (K.EQ.M2) B=DELF1
      IF (K.GT.M2) B=VV(I,KB+M1,97)
      DO 30 L=1,K1,M1
      LB=98-L
      Y=(L-1)*DEL
      EY=DEXP(-Y)
      EXY=EX*EY
      B=B-DELF1
      VN=0.D0
 31   VMAX=VN
      B=B+DELF1
      IF (B.GE.X) GO TO 32
      EB=DEXP(-B)
      AUX=B/DEL1
```

```
      IB=AUX
      BDIF=AUX-IB
      BDIF1=1.D0-BDIF
      IB=IB*M1+1
      IBXY=K+L-IB-M1
      IBX=K-IB-M1+1
      VN=(EB-EX-(X-B)*EXY)*(VV(J,IBX,L)*BDIF+VV(J,IBX+M1,L)*BDIF1)+
     &B*(EB-EXY)*(1.D0+V1(J,IBXY)*BDIF+V1(J,IBXY+M1)*BDIF1)+
     &(1.D0-(1.D0+B)*EB)*(VV(J,IB,1)*BDIF1+VV(J,IB+M1,1)*BDIF)
      IF (VN.GE.VMAX) GO TO 31
      B=B-DELF1
      GO TO 33
 32   KL=K+L-M2
      BMAX=X
C COARSE SEARCH FOR B.
      DO 35 M=K,KL,M1
      B=(M-1)*DEL
      EB=DEXP(-B)
      IBX=M-K+1
      IBXY=K+L-M
      VN=X*(EB-EXY)*(1.D0+V1(J,IBXY))+(1.D0-EX-X*EB)*VV(J,K,IBX)
      IF (VN.GE.VMAX) BMAX=B
 35   IF (VN.GE.VMAX) VMAX=VN
      BMAX1=BMAX
C FINE SEARCH FOR B.
      DO 36 M=1,19
      B=BMAX1-DEL1+M*DELF1
      IF (B.LT.X) GO TO 36
      EB=DEXP(-B)
      AUX=B/DEL1
      IB=AUX
      BDIF=AUX-IB
      BDIF1=1.D0-BDIF
      IB=IB*M1+1
      IBXY=K+L-IB-M1
      IBX=IB-K+1
```

```
      VN=X*(EB-EXY)*(1.D0+V1(J,IBXY)*BDIF+V1(J,IBXY+M1)*BDIF1)+
     &(1.D0-EX-X*EB)*(VV(J,K,IBX)*BDIF1+VV(J,K,IBX+M1)*BDIF)
      IF (VN.GE.VMAX) BMAX=B
      IF (VN.GE.VMAX) VMAX=VN
 36   CONTINUE
      B=BMAX
 33   VV(I,KB,LB)=B
      VV(I,K,L)=VMAX/(1.D0-EX-X*EXY)
      XX=VV(I,K,L)-VV(J,K,L)
      IF (XX.LT.XMIN) XMIN=XX
 30   IF (XX.GT.XMAX) XMAX=XX
      B=VV(I,2,97)
C CALCULATE V2(X) AND B(X), X>.96.
      DO 40 L=1,51
      X=DEL*(L-1)+.97D0
      EX=DEXP(-X)
      B=B-DELF
      VN=0.D0
 41   VMAX=VN
      B=B+DELF
      EB=DEXP(-B)
      AUX=B/DEL1
      IB=AUX
      BDIF=AUX-IB
      BDIF1=1.D0-BDIF
      IB=IB*M1+1
      AUX=(X-B)/DEL1
      IBX=AUX
      BXDIF=AUX-IBX
      BXDIF1=1.D0-BXDIF
      IBX=IBX*M1+1
      VN=(EB-(1.D0+X-B)*EX)*(VV(J,IBX,1)*BXDIF1+VV(J,IBX+M1,1)*BXDIF)+
     &B*(EB-EX)*(1.D0+V1(J,IBX)*BXDIF1+V1(J,IBX+M1)*BXDIF)+
     &(1.D0-(1.D0+B)*EB)*(VV(J,IB,1)*BDIF1+VV(J,IB+M1,1)*BDIF)
      IF (VN.GE.VMAX) GO TO 41
      B=B-DELF
```

```
      BB(L)=B
      V2(I,L)=VMAX/(1.D0-(1.D0+X)*EX)
      X=V2(I,L)-V2(J,L)
      IF (X.LT.XMIN) XMIN=X
 40   IF (X.GT.XMAX) XMAX=X
C PRINT CONTENTS OF MATRICES.
      WRITE (6,50) N,T,XMIN,XMAX
 50   FORMAT ('1',I4,4X,F8.5,4X,F8.5,4X,F8.5)
      WRITE (6,101)
 101  FORMAT (' ')
      WRITE (6,102) V0(I)
 102  FORMAT (' ',4X,17(1X,F6.3))
      WRITE (6,101)
      WRITE (6,102) (PRIN(L),L=1,97,6)
      WRITE (6,102) (V1(I,L),L=1,97,6)
      WRITE (6,102) (AA(L),L=1,97,6)
      WRITE (6,101)
      WRITE (6,102) (PRIN(L),L=1,97,6)
      DO 99 K=1,97,6
      K1=98-K
      WRITE (6,103) PRIN(K),(VV(I,K,L),L=1,K1,6)
 99   CONTINUE
 103  FORMAT (' ',F4.2,17(1X,F6.3))
      WRITE (6,101)
      WRITE (6,102) (PRIN(L),L=1,97,6)
      DO 100 K=1,97,6
      K1=98-K
      K2=99-K
      DO 110 L=K,97,6
      LL=98-L
 110  PB(LL)=VV(I,K2,L)
      WRITE (6,103) PRIN(K),(PB(L),L=1,K1,6)
 100  CONTINUE
      WRITE (6,101)
      WRITE (6,102) (V2(I,L),L=1,17)
      WRITE (6,102) (BB(L),L=1,17)
```

```
      WRITE (6,101)
      WRITE (6,102) (V2(I,L),L=18,34)
      WRITE (6,102) (BB(L),L=18,34)
      WRITE (6,101)
      WRITE (6,102) (V2(I,L),L=35,51)
      WRITE (6,102) (BB(L),L=35,51)
      XDIF=XMAX-XMIN
      IF (XDIF.LT..00001) GO TO 2
1     CONTINUE
2     STOP
      END
```

```
C PROGRAM TO CALCULATE V0, V1(X) AND V2(X,Y) USING A SMALL, FINE GRID.
      IMPLICIT REAL*8 (A-H,O-$)
      DIMENSION V0(2),V1(2,80),V2(2,130),VV(2,16,31)
      DIMENSION AA(80),BB(130),PRIN(130),VB(16,31)
      DEL=.01D0
      DELF=.001D0
C INITIALIZE CONTENTS OF MATRICES.
      V1(1,1)=-1.D0
      BB(1)=0.D0
      V0(2)=0.D0
      DO 6 I=1,80
 6    V1(2,I)=0.D0
      DO 7 I=1,130
      PRIN(I)=(I-1)*DEL
 7    V2(2,I)=0.D0
      DO 8 J=1,31
      VB(1,J)=0.D0
      DO 8 I=1,16
 8    VV(2,I,J)=0.D0
      DO 1 N=1,25
      I=2-(N-(N/2)*2)
      J=1+N-(N/2)*2
C CALCULATE V0 AND T.
      VMAX=0.D0
      TMAX1=.99D0
C COARSE SEARCH FOR T.
      DO 10 L=1,29
      T=TMAX1+L*DEL
      TEX=DEXP(-T)
      IT=100+L
      VN=T*TEX+(1.D0+T)*TEX*V0(J)+V2(J,IT)*(1.D0-(1.D0+T)*TEX)
      IF (VN.GE.VMAX) TMAX=T
 10   IF (VN.GE.VMAX) VMAX=VN
C FINE SEARCH FOR T.
      DO 11 L=1,19
```

```
      T=TMAX-.01D0+L*DELF
      TEX=DEXP(-T)
      AUX=T/DEL
      IT=AUX
      TDIF=AUX-IT
      TDIF1=1.D0-TDIF
      IT=IT+1
      VN=T*TEX+(1.D0+T)*TEX*V0(J)+(V2(J,IT)*TDIF1+V2(J,IT+1)*TDIF)
     &*(1.D0-(1.D0+T)*TEX)
      IF (VN.GE.VMAX) TMAX1=T
 11   IF (VN.GE.VMAX) VMAX=VN
      V0(I)=VMAX
      T=TMAX1
      X=V0(I)-V0(J)
      XMIN=X
      XMAX=X
C CALCULATE V1(0) AND A(0).
      VMAX=0.D0
C COARSE SEARCH FOR A.
      DO 15 L=1,30
      A=(L-1)*DEL
      EA=DEXP(-A)
      VN=EA*(1.D0+V0(J))+(1.D0-EA)*VV(J,1,L)
      IF (VN.GE.VMAX) AMAX1=A
 15   IF (VN.GE.VMAX) VMAX=VN
      AMAX=AMAX1
C FINE SEARCH FOR A.
      DO 16 L=1,19
      A=AMAX1-DEL+L*DELF
      IF (A.LT.0) GO TO 16
      EA=DEXP(-A)
      AUX=A/DEL
      IA=AUX
      ADIF=AUX-IA
      ADIF1=1.D0-ADIF
      VN=EA*(1.D0+V0(J))+(1.D0-EA)*VV(J,1,IA+1)*ADIF
```

```
      IF (IA.EQ.1) VN=VN+(1.DO-EA)*(1.DO+V1(I,1))*ADIF1
      IF (IA.GT.1) VN=VN+(1.DO-EA)*VV(J,1,IA)*ADIF1
      IF (VN.GE.VMAX) AMAX=A
      IF (VN.GE.VMAX) VMAX=VN
   16 CONTINUE
      AA(1)=AMAX
      V1(I,1)=VMAX
      X=V1(I,1)-V1(J,1)
      IF (X.LT.XMIN) XMIN=X
      IF (X.GT.XMAX) XMAX=X
C CALCULATE V1(X) AND A(X),0<X<.16.
C THESE ARE THE VALUES OF X WHERE A MAY BE > X.
      DO 20 L=2,16
      X=(L-1)*DEL
      EX=DEXP(-X)
      VMAX=0.DO
C COARSE SEARCH FOR A.
      DO 21 LA=1,30
      A=X+(LA-1)*DEL
      EA=DEXP(-A)
      VN=X*EA*(1.DO+VO(J))+(1.DO-EX-X*EA)*VV(J,L,LA)
      IF (VN.GE.VMAX) AMAX1=A
   21 IF (VN.GE.VMAX) VMAX=VN
      AMAX=AMAX1
C FINE SEARCH FOR A.
      DO 24 LA=1,19
      A=AMAX1-DEL+LA*DELF
      IF (A.LT.X) GO TO 24
      EA=DEXP(-A)
      AUX=(A-X)/DEL
      IAX=AUX
      AXDIF=AUX-IAX
      AXDIF1=1.DO-AXDIF
      IAX=IAX+1
      VN=X*EA*(1.DO+VO(J))+(1.DO-EX-X*EA)*(VV(J,L,IAX)*AXDIF1
     &+VV(J,L,IAX+1)*AXDIF)
```

```
      IF (VN.GE.VMAX) AMAX=A
      IF (VN.GE.VMAX) VMAX=VN
 24   CONTINUE
      AA(L)=AMAX
      V1(I,L)=VMAX/(1.D0-EX)
      X=V1(I,L)-V1(J,L)
      IF (X.LT.XMIN) XMIN=X
 20   IF (X.GT.XMAX) XMAX=X
C CALCULATE V1(X) AND A(X), X>.15.
C THESE ARE THE VALUES OF X WHERE A MUST BE <X.
      A=.15D0
      DO 60 L=17,80
      X=(L-1)*DEL
      EX=DEXP(-X)
      A=A-DELF
      VN=0.D0
 61   VMAX=VN
      A=A+DELF
      IF (A.GE.X) GO TO 62
      EA=DEXP(-A)
      AUX=A/DEL
      IA=AUX
      ADIF=AUX-IA
      ADIF1=1.D0-ADIF
      IA=IA+1
      IAX=L-IA
      VN=(EA-EX)*(V1(J,IAX)*ADIF+V1(J,IAX+1)*ADIF1)+A*EA*(1.D0+V0(J))+
     &(1.D0-(1.D0+A)*EA)*(V2(J,IA)*ADIF1+V2(J,IA+1)*ADIF)
      IF (VN.GE.VMAX) GO TO 61
      A=A-DELF
      GO TO 63
 62   VN=X*EX*(1.D0+V0(J))+(1.D0-(1.D0+X)*EX)*V2(J,L)
      A=A-DELF
      IF (VN.GE.VMAX) A=X
      IF (VN.GE.VMAX) VMAX=VN
 63   AA(L)=A
```

```
      V1(I,L)=VMAX/(1.D0-EX)
      X=V1(I,L)-V1(J,L)
      IF (X.LT.XMIN) XMIN=X
 60   IF (X.GT.XMAX) XMAX=X
C CALCULATE VV(0,0).
      VV(I,1,1)=(VV(J,1,1)+V1(J,1)+1.D0)/2.D0
      V2(I,1)=VV(I,1,1)
      X=VV(I,1,1)-VV(J,1,1)
      IF (X.LT.XMIN) XMIN=X
      IF (X.GT.XMAX) XMAX=X
C CALCULATE VV(0,Y), Y>0.
      DO 29 L=2,31
      VV(I,1,L)=1.D0+V1(J,L)
      X=VV(I,1,L)-VV(J,1,L)
      IF (X.LT.XMIN) XMIN=X
 29   IF (X.GT.XMAX) XMAX=X
C CALCULATE VV(X,Y) AND B(X,Y), X>0.
      DO 30 K=2,16
      X=(K-1)*DEL
      EX=DEXP(-X)
      IF (K.EQ.2) B=DELF
      IF (K.GT.2) B=VB(K-1,1)
      DO 30 L=1,31
      Y=(L-1)*DEL
      EY=DEXP(-Y)
      EXY=EX*EY
      B=B-DELF
      VN=0.D0
 31   VMAX=VN
      B=B+DELF
      IF (B.GE.X) GO TO 32
      EB=DEXP(-B)
      AUX=B/DEL
      IB=AUX
      BDIF=AUX-IB
      BDIF1=1.D0-BDIF
```

```
      IB=IB+1
      IBXY=K+L-IB-1
      IBX=K-IB
      VN=(EB-EX-(X-B)*EXY)*(VV(J,IBX,L)*BDIF+VV(J,IBX+1,L)*BDIF1)+
     &B*(EB-EXY)*(1.D0+V1(J,IBXY)*BDIF+V1(J,IBXY+1)*BDIF1)+
     &(1.D0-(1.D0+B)*EB)*(V2(J,IB)*BDIF1+V2(J,IB+1)*BDIF)
      IF (VN.GE.VMAX) GO TO 31
      B=B-DELF
      GO TO 33
   32 VN=X*(EX-EXY)*(1.D0+V1(J,L))+(1.D0-(1.D0+X)*EX)*V2(J,K)
      B=B-DELF
      IF (VN.GE.VMAX) B=X
      IF (VN.GE.VMAX) VMAX=VN
   33 VB(K,L)=B
      VV(I,K,L)=VMAX/(1.D0-EX-X*EXY)
      IF (L.EQ.1) V2(I,K)=VV(I,K,L)
      IF (L.EQ.1) BB(K)=VB(K,L)
      XX=VV(I,K,L)-VV(J,K,L)
      IF (XX.LT.XMIN) XMIN=XX
   30 IF (XX.GT.XMAX) XMAX=XX
      B=VB(16,1)
C CALCULATE VV(X,0) AND B(X), X>.15.
      DO 40 L=17,130
      X=DEL*(L-1)
      EX=DEXP(-X)
      B=B-DELF
      VN=0.D0
   41 VMAX=VN
      B=B+DELF
      EB=DEXP(-B)
      AUX=B/DEL
      IB=AUX
      BDIF=AUX-IB
      BDIF1=1.D0-BDIF
      IB=IB+1
      IBX=L-IB
```

```
      VN=(EB-(1.D0+X-B)*EX)*(V2(J,IBX)*BDIF+V2(J,IBX+1)*BDIF1)+
     &B*(EB-EX)*(1.D0+V1(J,IBX)*BDIF+V1(J,IBX+1)*BDIF1)+
     &(1.D0-(1.D0+B)*EB)*(V2(J,IB)*BDIF1+V2(J,IB+1)*BDIF)
      IF (VN.GE.VMAX) GO TO 41
      B=B-DELF
      BB(L)=B
      V2(J,L)=VMAX/(1.D0-(1.D0+X)*EX)
      X=V2(I,L)-V2(J,L)
      IF (X.LT.XMIN) XMIN=X
 40   IF (X.GT.XMAX) XMAX=X
C PRINT CONTENTS OF MATRICES.
      WRITE (6,50) N,T,XMIN,XMAX
 50   FORMAT ('1',I4,4X,F8.5,4X,F8.5,4X,F8.5)
      WRITE (6,101)
 101  FORMAT (' ')
      WRITE (6,102) VC(I)
 102  FORMAT (' ',4X,16(1X,F6.3))
      WRITE (6,101)
      WRITE (6,102) (PRIN(L),L=1,16)
      WRITE (6,102) (V1(I,L),L=1,16)
      WRITE (6,102) (AA(L),L=1,16)
      WRITE (6,101)
      WRITE (6,102) (PRIN(L),L=17,32)
      WRITE (6,102) (V1(I,L),L=17,32)
      WRITE (6,102) (AA(L),L=17,32)
      WRITE (6,101)
      WRITE (6,102) (PRIN(L),L=33,48)
      WRITE (6,102) (V1(I,L),L=33,48)
      WRITE (6,102) (AA(L),L=33,48)
      WRITE (6,101)
      WRITE (6,102) (PRIN(L),L=49,64)
      WRITE (6,102) (V1(I,L),L=49,64)
      WRITE (6,102) (AA(L),L=49,64)
      WRITE (6,101)
      WRITE (6,102) (PRIN(L),L=65,80)
      WRITE (6,102) (V1(I,L),L=65,80)
```

```
      WRITE (6,102) (AA(L),L=65,80)
      WRITE (6,101)
      WRITE (6,102) (PRIN(L),I=1,16)
      DO 99 K=1,31
      WRITE (6,103) PRIN(K),(VV(I,L,K),L=1,16)
99    CONTINUE
103   FORMAT (' ',F4.2,16(1X,F6.3))
      WRITE (6,101)
      WRITE (6,102) (PRIN(L),L=1,16)
      DO 100 K=1,31
      WRITE (6,103) PRIN(K),(VB(L,K),L=1,16)
100   CONTINUE
      WRITE (6,101)
      WRITE (6,102) (PRIN(L),L=1,16)
      WRITE (6,102) (V2(I,L),L=1,16)
      WRITE (6,102) (BB(L),L=1,16)
      WRITE (6,101)
      WRITE (6,102) (PRIN(L),L=17,32)
      WRITE (6,102) (V2(I,L),L=17,32)
      WRITE (6,102) (BB(L),L=17,32)
      WRITE (6,101)
      WRITE (6,102) (PRIN(L),I=33,48)
      WRITE (6,102) (V2(I,L),L=33,48)
      WRITE (6,102) (BB(L),L=33,48)
      WRITE (6,101)
      WRITE (6,102) (PRIN(L),L=49,64)
      WRITE (6,102) (V2(I,L),L=49,64)
      WRITE (6,102) (BB(L),L=49,64)
      WRITE (6,101)
      WRITE (6,102) (PRIN(L),L=65,80)
      WRITE (6,102) (V2(I,L),L=65,80)
      WRITE (6,102) (BB(L),L=65,80)
      WRITE (6,101)
      WRITE (6,102) (PRIN(L),L=81,96)
      WRITE (6,102) (V2(I,L),L=81,96)
      WRITE (6,102) (BB(L),L=81,96)
```

```
      WRITE (6,101)
      WRITE (6,102) (PRIN(L),L=97,112)
      WRITE (6,102) (V2(I,L),L=97,112)
      WRITE (6,102) (BB(L),L=97,112)
      WRITE (6,101)
      WRITE (6,102) (PRIN(L),L=113,128)
      WRITE (6,102) (V2(I,L),L=113,128)
      WRITE (6,102) (BB(L),L=113,128)
      WRITE (6,101)
      WRITE (6,102) (PRIN(L),L=129,130)
      WRITE (6,102) (V2(I,L),L=129,130)
      WRITE (6,102) (BB(L),L=129,130)
      XDIF=XMAX-XMIN
      IF (XDIF.LT..00001) GO TO 2
1     CONTINUE
2     STOP
      END
```

REFERENCES

1. Abramson, N., Packet Switching with Satellites, National Computer Conference, Vol. 42, 1973.

2. Roberts, Lawrence G., Aloha Packet System with and without Slots and Capture, ASS, Note 8, June 26, 1972.

3. Metcalfe,Robert M., Packet Communication, M.I.T., MAC TR-114, December 1973.

4. Roberts, L. G., Dynamic Allocation of Satellite Capacity through Packet Reservation, 1973 Nat. Comput. Conf., AFIPS Conf. Proc., vol. 42.

5. Kleinrock, L. and Tobagi, F., Carrier-sense Multiple Access for Packet Switched Radio Channels, Proc. Int. Conf. Communications, Minneapolis, Minn., June 1974.

6. Capetanakis, J., The Multiple Access Broadcast Channel: Protocol and Capacity Considerations, Ph.D. Thesis, M.I.T., 1977.

7. Gallager, R. G., A New Slotted Random Access Channel Control Strategy, unpublished paper.

8. Humblet, P., private communication.

9. Howard, R., Dynamic Programming and Markov Processes, Cambridge, MA, M.I.T. Press, 1960.

10. Scweitzer, P. J., Perturbation Theory and Markovian Decision Processes, MIT Operations Research Center Technical Report No. 15, June 1965.

11. Odoni, A. R., On Finding the Maximal Gain for Markov Decision Processes, Operations Research, vol. 17, No. 5, Sept-Oct, 1969.

Distribution List

Defense Documentation Center Cameron Station Alexandria, Virginia 22314	12 Copies
Assistant Chief for Technology Office of Naval Research, Code 200 Arlington, Virginia 22217	1 Copy
Office of Naval Research Information Systems Program Code 437 Arlington, Virginia 22217	2 Copies
Office of Naval Research Branch Office, Boston 495 Summer Street Boston, Massachusetts 02210	1 Copy
Office of Naval Research Branch Office, Chicago 536 South Clark Street Chicago, Illinois 60605	1 Copy
Office of Naval Research Branch Office, Pasadena 1030 East Greet Street Pasadena, California 91106	1 Copy
New York Area Office (ONR) 715 Broadway - 5th Floor New York, New York 10003	1 Copy
Naval Research Laboratory Technical Information Division, Code 2627 Washington, D.C. 20375	6 Copies
Dr. A. L. Slafkosky Scientific Advisor Commandant of the Marine Corps (Code RD-1) Washington, D.C. 20380	1 Copy

Office of Naval Research
Code 455
Arlington, Virginia 22217
1 Copy

Office of Naval Research
Code 458
Arlington, Virginia 22217
1 Copy

Naval Electronics Laboratory Center
Advanced Software Technology Division
Code 5200
San Diego, California 92152
1 Copy

Mr. E. H. Gleissner
Naval Ship Research & Development Center
Computation and Mathematics Department
Bethesda, Maryland 20084
1 Copy

Captain Grace M. Hopper
NAICOM/MIS Planning Branch (OP-916D)
Office of Chief of Naval Operations
Washington, D.C. 20350
1 Copy

Mr. Kin B. Thompson
Technical Director
Information Systems Division (OP-91T)
Office of Chief of Naval Operations
Washington, D.C. 20350
1 Copy

Advanced Research Projects Agency
Information Processing Techniques
1400 Wilson Boulevard
Arlington, Virginia 22209
1 Copy

Dr. Stuart L. Brodsky
Office of Naval Research
Code 432
Arlington, Virginia 22217
1 Copy

Captain Richard L. Martin, USN
Commanding Officer
USS Francis Marion (LPA-249)
FPO New York 09501
1 Copy